

TURBOCHARGED EMC NETWORKER

v1.1 | April 2015
nsrd.info

About the author



Preston de Guise is an experienced data protection consultant who has been working in the field for almost two decades with a focus throughout most of that career in enterprise level data backup and recovery. He has been using EMC (and formerly Legato) NetWorker for 19 years.

Preston is the author of *Enterprise Systems Backup and Recovery: A Corporate Insurance Policy*, CRC Press (2008, ISBN-13 978-1420076394), and is currently working on a new book, *Data Protection: Preventing Data Loss in the Age of Big Data, Cloud, and Virtualization*.

Preston may be contacted via email to preston@nsrd.info.

About this manual

This manual is provided free of charge from the NetWorker Information Hub (<http://nsrd.info>).

You may use the manual as you wish, but you must not:

- Modify it;
- Claim any content within it as your own;
- Sell or exert a copyright claim over any script derived from scripts contained within the manual or linked to from the manual;
- Redistribute it without permission from the author.

The instructions provided in this manual may, if used incorrectly or without consideration, cause problems or data loss within your environment. No warranty is made whatsoever by the author as to the applicability of commands and instructions for your environment, and the author is not responsible or liable for any issues or data loss that may occur as a result of following the instructions in this manual.

Document History

Version	Released	Description
1.1	April, 2015	Added mminfo reporting options for VBA backups. Added 'nsradmin -C' Added information about dbgcommand. Minor corrections. Moved tables of figures and index of tables to end of document.
1.0	January, 2015	First release

Corrections or questions regarding this document can be directed to preston@nsrd.info.

Contents

INTRODUCTION	7
1 INTENDED AUDIENCE	7
2 HOW THIS GUIDE SHOULD BE USED	8
3 KEEP THE MAN PAGES HANDY	8
4 TYPES OF NETWORKER COMMANDS	8
5 CONVENTIONS USED	9
6 SCRIPTING EXAMPLES	9
REPORTING COMMANDS	10
7 INTRODUCTION	10
8 MMINFO	11
8.1 Ordering the Results	12
8.2 Specifying the Query	12
8.2.1 Specifying Multiple Queries	16
8.2.2 Many query options	17
8.3 Media Queries	19
8.4 Specifying the Report Columns	20
8.5 Enhancements for VBA Backups	25
8.6 XM your L	26
8.7 Stepping it up with a scripting language	29
8.7.1 Example: Script to determine order of media to scan	29
8.7.2 Example: mminfo2html	34
9 NSRINFO	36
9.1 Command Line Arguments	36
9.2 Listing files by time	37
9.3 Finding previously backed up files	38
10 GSTCLREPORT	41
11 MISCELLANEOUS REPORTING	47
11.1 nsr_render_log	47
11.1.1 Remote rendering	49
11.1.2 Aside – Auto-rendered Log Files	50
11.2 nsrgrpcomp	52
11.3 Using nsrwatch	56
CONTROL COMMANDS	60
12 INTRODUCTION	60
13 NSRMM	61
13.1 What is nsrmm?	61
13.2 Warning	61

13.3	Lab Environment	61
13.4	Media manipulation	61
13.5	Media Database manipulation	65
13.6	Changing the mode of a volume or saveset	65
13.7	Changing Browse/Retention Time of a Saveset	68
14	TAPE LIBRARY OPERATIONS	69
14.1	nsrjb	69
14.1.1	Showing the jukebox contents	72
14.1.2	Jukebox Inventory	72
14.1.3	Resetting a Jukebox	76
14.1.4	Labelling and Relabeling Media	77
14.1.5	Loading and Unloading Volumes	79
14.1.6	Exporting and Importing Media	81
14.2	Low level interaction	83
14.2.1	sjisn	84
14.2.2	sjirdtag	85
14.2.3	sjimm	87
15	NSRADMIN	89
15.1	Warning	89
15.2	Getting Started	89
15.3	Offline vs Online	90
15.4	Your Lab Environment	91
15.5	Running nsradmin	91
15.6	Syntax Overview	92
15.7	Starting and Stopping Backups	99
15.7.1	What you'll need	99
15.7.2	Monitoring	100
15.7.3	Starting a Backup	100
15.7.4	Stopping a Running Backup	101
15.8	Checking the Status of a Group	103
15.8.1	Cloning and Monitoring	104
15.9	Append vs Update	105
15.10	Setting up regular backup components	106
15.10.1	Browse and Retention Policies	107
15.10.2	Schedules	109
15.10.3	Groups	113
15.10.4	Clients	115
15.10.5	Pools	117
15.10.6	Revisiting the Groups	121
15.11	Monitoring Devices	122
15.12	Deleting Resources	124
15.13	Bulk Commands	128
15.14	Scripting	131
15.14.1	Introductory Scripting	132
15.14.2	Setting up for Scripted Client Creation	133

15.14.3	A client creation script	135
15.15	Connecting to Client Services	137
15.16	Using regular expressions in nsradmin	139
15.17	Offline mode	141
MAINTENANCE		143
<hr/>		
16	INTRODUCTION	143
17	HEALTH CHECK COMMANDS	144
17.1	Media Database Check	144
17.2	Index Checks	145
17.3	Index Management	148
18	CLIENT CONNECTIVITY CHECKING	149
19	USING DBGCOMMAND	151
19.1	Correlating devices to running daemons	152
19.2	Flushing NetWorker's Internal DNS Cache	154
19.3	Turning on Debug Mode	154
20	LOG MAINTENANCE	155
BACKUP CONTROL		157
<hr/>		
21	INTRODUCTION	157
22	PRE AND POST PROCESSING COMMANDS	158
22.1	Advantages of pre and post processing	158
22.2	savenpc	158
22.3	The new order	159
23	NETWORKER DIRECTIVES	163
23.1	Overview	163
23.2	Placement	165
23.3	Directive Examples	166
23.3.1	Scenario: Skipping Database files on Microsoft SQL Server	166
23.3.2	Example: Skipping Multimedia Content	167
23.3.3	Example: Split Backups of a very large filesystem	168
WRAPPING UP		172
<hr/>		
FURTHER READING		173
<hr/>		
INDICES		175
<hr/>		

Introduction

1 Intended Audience

This guide is targeted at people who have been NetWorker administrators for at least 6-12 months. Some of the concepts outlined within the manual define processes that, if used incorrectly, or used against a production server without suitable modification, may cause issues.

As such, it's recommended the reader be reasonably familiar with the process and operation of EMC NetWorker before reading this guide.

The processes and instructions in this guide focus mostly on working with NetWorker from the command line. In instances where there is an obvious advantage to working within a GUI, this may be stated, but wherever possible, the command line process will be provided.

Remember, the most important thing about a command line interface is that it is fully automatable. Automation is, by far and away, a key aspect of being a power-user for any product.

2 How this guide should be used

Some sections and examples used in this guide are expressly oriented towards running in a lab environment, away from production systems. Almost all reporting examples within the guide will be *site-specific*; i.e., the output or results received will be entirely dependent on the environment they are run in. A report that may generate only a few lines in this guide, for instance, may generate hundreds or even hundreds of thousands of lines of output on a production system.

Commands and instructions will be broken into two categories for the guide:

- **Production-ready** – able to be run immediately in a production environment, with the exception of appropriate modifications for host names, passwords, local date formats, etc.
- **Lab-only** – a command that should only be run against a lab environment until you are 100% confident that you understand the results of the actions and you have appropriate backups in place.

The author takes no responsibility for any scenario where a command executed as described causes an issue; in all instances, an administrator is ultimately responsible for the commands and actions he or she takes in a NetWorker environment.

3 Keep the man pages handy

On Unix/Linux systems at least, the NetWorker command line comes with extensive help via the *man* pages. (E.g., “man mminfo”). If you’re using Windows, don’t despair – help comes in a slightly different format. The PDF documentation for NetWorker includes a *Command Reference Guide*, which is comprised of the individual man pages for all NetWorker CLI options.

The man pages should be your constant companion when working with NetWorker from the command line, regardless of whether you access them via *man* on Unix/Linux, or whether you have the *Command Reference Guide* PDF open at the same time as your command window.

4 Types of NetWorker Commands

There are three key categories of commands one might use in EMC NetWorker:

- **Control** – These include backup, recovery, configuration manipulation and data processing.
- **Maintenance** – These are tasks that you should be periodically performing (or comfortable performing) on your NetWorker server.
- **Reporting** – These provide access to the media database, file indices and NetWorker Management Console database.

As you might gather, *control* functions are ones to be particularly mindful of when running within a production environment. By and large, you should assume that *all* control functions described in the guide are intended for use first against a lab environment for familiarisation, before being *adapted* for use as required within a production environment.

Maintenance tasks are the basis for ensuring the NetWorker environment is running in peak condition. These include various checks that you can run, and tasks that you should run before performing upgrades or changes to the NetWorker environment. As per control functions, you should make yourself comfortable running the commands within a lab environment first before considering applying them, suitably adapted, to your own production environment.

Reporting typically should not cause a problem within a production environment, with the obvious caveats that requesting *too much* data may cause a momentary spike in NetWorker or NMC Server resources, and any system errors or corruption present within an environment can cause any tool to behave erratically.

5 Conventions Used

Before any set of instructions that, if misapplied, could cause an issue, the following will be noted:

CAUTION - Lab Exercise

Be certain to check all instructions for this warning. If uncertain, always run a command in the lab first.

The only exception to the above is where an entire *topic* covers information that, if misused, could cause damage to your NetWorker environment. These topics will be prefaced with a *Warning* section that should be followed carefully.

Commands that you enter will be presented in fixed-width font. Prompts will be shown in regular weight text, and the actual commands you enter will be shown in bold. For example:

```
nsradmin> print type: NSR client
```

This implies the prompt was “nsradmin>” and the entered text was “print type: NSR client”.

6 Scripting Examples

The *Perl* programming language is used for most scripting examples in this guide, though some smaller scripts will be presented in Windows batch format as well. If you’re not familiar with Perl, don’t worry – the examples provided are self-contained and do not need modification to function in a local environment.

Perl is installed by default on most Unix/Linux systems, and can be installed on Windows via third party providers, such as ActiveState (www.activestate.com).

Reporting Commands

7 Introduction

Within the reporting commands, there's an obvious gorilla in the room – **mminfo**. Yet, this is not whole story when it comes to NetWorker reporting.

In this chapter, we'll focus on the three essential reporting tools available to a NetWorker administrator:

- **mminfo** – Interrogates the media database
- **nsrinfo** – Interrogates client indices
- **gstclreport** – Provides access to NMC reporting

Additionally, we'll look at:

- **nsr_render_log** – Provides a human readable version of a '.raw' log file
- **nsrsgrpcomp** – Accesses and displays recent savegroup completion information

In particular, it's worth noting that many of the reporting capabilities in NetWorker are best experimented with regularly. As you use them more often, and try additional options, you'll discover new ways of getting helpful information out of the product.

8 mminfo

To quote the NetWorker man/help page for mminfo, this utility is the “NetWorker media database reporting command”. It should be an essential feature in any NetWorker administrator's toolkit.

Let's start with the basics. Run without any arguments, mminfo reports all *successfully completed* backups performed in the last 24 hours:

```
# mminfo

[root@orilla ~]# mminfo
volume      client      date        size        level name
Slow.01     miranda     19/07/14    1626 MB     incr /
Slow.01     miranda     19/07/14    216 MB     incr /
Slow.01     mondas      19/07/14    9251 KB     incr /
Slow.01     mondas      19/07/14    4 B        incr /boot
Slow.01     mondas      19/07/14    4 B        incr /d/01
Slow.01     mondas      19/07/14    2 KB       incr /d/backup1
Slow.01     orilla.turbamentis.int 19/07/14    7206 KB     incr /
Slow.01     orilla.turbamentis.int 19/07/14    4 B        incr /synology/finance
Slow.01     orilla.turbamentis.int 19/07/14    1077 KB     incr /synology/pmdg/Documents
Slow.01     orilla.turbamentis.int 19/07/14    181 KB     full CONSOLE_BACKUP_FILES
Slow.01     orilla.turbamentis.int 20/07/14    181 KB     full CONSOLE_BACKUP_FILES
Slow.01     orilla.turbamentis.int 19/07/14    6145 KB     full NMCASA:/gst_on_orilla/lgto_gst
Slow.01     orilla.turbamentis.int 20/07/14    6257 KB     full NMCASA:/gst_on_orilla/lgto_gst
Staging.02  hyperion    19/07/14    85 MB      incr /Volumes/Alteran/Documents/Music
Standard.01 archon      19/07/14    3879 MB     incr /
Standard.01 archon      19/07/14    3052 MB     incr /
Standard.01 archon      19/07/14    46 KB       incr /Volumes/Encrypted
Standard.01 archon      19/07/14    11 KB       incr /Volumes/Encrypted
```

Figure 1: mminfo default output (24 hours backups)

If you're not familiar with mminfo's output conventions, the sort order may need explaining. The *default* sort order is:

- media
- offset on media
- client
- name
- time
- length of this part of the saveset

This output format is highly suitable for tape based backups, but less so for modern backup targets such as Data Domain and Advanced File Type devices.

8.1 Ordering the Results

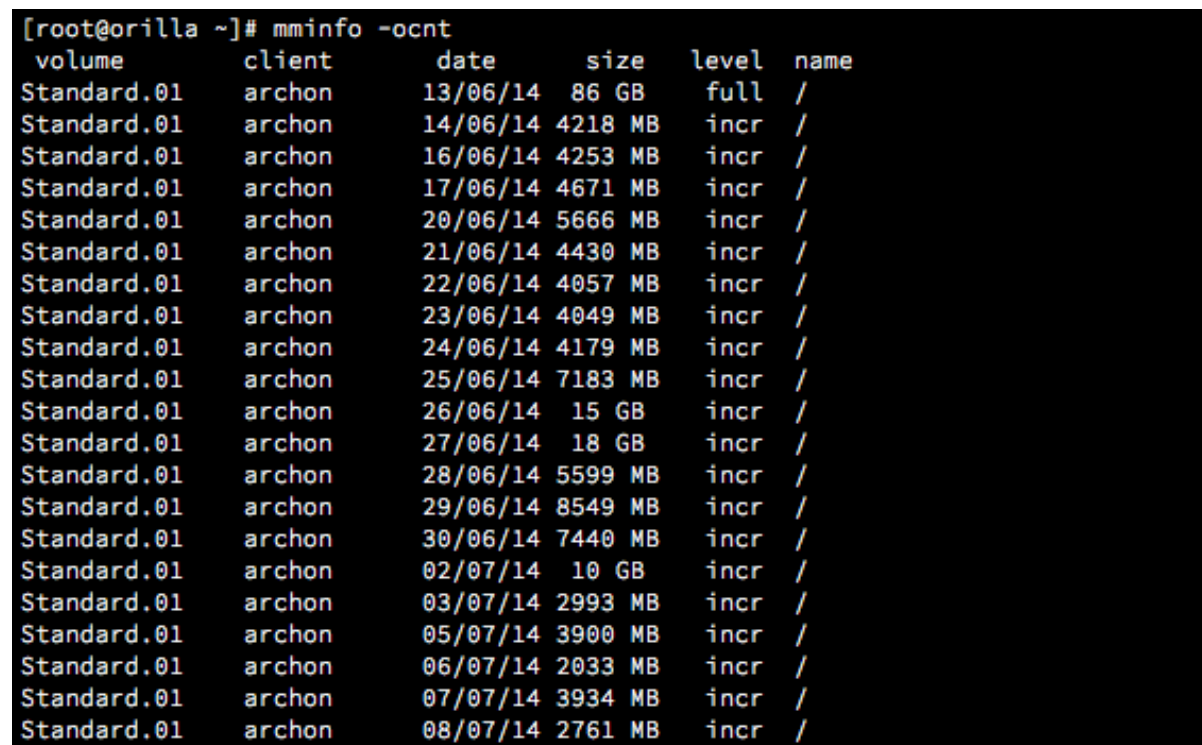
The sort order for `mminfo` output can be changed by using the `-o` parameter. This parameter takes the following options:

Option	Meaning
c	Client
e	Expiration date
l	Length
m	Media name
n	Name of the client
o	Offset of the backup from the start of the media
t	Timestamp for the saveset
R	Reverse the output ordering

Table 1: Ordering options in `mminfo`

For example, to order `mminfo` output by client, saveset name and then time, one would use `-ocnt`:

```
# mminfo -ocnt
```



volume	client	date	size	level	name
Standard.01	archon	13/06/14	86 GB	full	/
Standard.01	archon	14/06/14	4218 MB	incr	/
Standard.01	archon	16/06/14	4253 MB	incr	/
Standard.01	archon	17/06/14	4671 MB	incr	/
Standard.01	archon	20/06/14	5666 MB	incr	/
Standard.01	archon	21/06/14	4430 MB	incr	/
Standard.01	archon	22/06/14	4057 MB	incr	/
Standard.01	archon	23/06/14	4049 MB	incr	/
Standard.01	archon	24/06/14	4179 MB	incr	/
Standard.01	archon	25/06/14	7183 MB	incr	/
Standard.01	archon	26/06/14	15 GB	incr	/
Standard.01	archon	27/06/14	18 GB	incr	/
Standard.01	archon	28/06/14	5599 MB	incr	/
Standard.01	archon	29/06/14	8549 MB	incr	/
Standard.01	archon	30/06/14	7440 MB	incr	/
Standard.01	archon	02/07/14	10 GB	incr	/
Standard.01	archon	03/07/14	2993 MB	incr	/
Standard.01	archon	05/07/14	3900 MB	incr	/
Standard.01	archon	06/07/14	2033 MB	incr	/
Standard.01	archon	07/07/14	3934 MB	incr	/
Standard.01	archon	08/07/14	2761 MB	incr	/

Figure 2: Specifying an alternate order output for `mminfo`

You may note that this is giving substantially more/different output from the previous command. As stated, when run *without any arguments*, `mminfo` will report all successfully completed backups generated in the last 24 hours.

However, even specifying the *order* of the output triggers `mminfo` to behave differently – and that’s to provide information on *all* backups.

8.2 Specifying the Query

To start, let’s return the previous query to the *default* query – everything backed up in the last 24 hours:

```
# mminfo -q "savetime>=24 hours ago" -ocnt
```

```
[root@orilla ~]# mminfo -q "savetime>=24 hours ago" -ocnt
volume      client      date       size      level      name
Standard.01 archon      19/07/14  3879 MB   incr       /
Standard.01 archon      19/07/14  3052 MB   incr       /
Standard.01 archon      19/07/14   46 KB    incr       /Volumes/Encrypted
Standard.01 archon      19/07/14   11 KB    incr       /Volumes/Encrypted
Standard.01 archon      19/07/14  9940 KB   incr       /Volumes/Storage
Standard.01 archon      19/07/14   13 KB    incr       /Volumes/Storage
Standard.01 hyperion    19/07/14  4656 MB   incr       /
Staging.02  hyperion    19/07/14   85 MB    incr       /Volumes/Alteran/Documents/M
usic
Standard.01 hyperion    19/07/14    5 KB     incr       /Volumes/Promise/Media
Slow.01     miranda     19/07/14  1626 MB   incr       /
Slow.01     miranda     19/07/14   216 MB    incr       /
Slow.01     mondas      19/07/14  9251 KB   incr       /
Slow.01     mondas      19/07/14    4 B       incr       /boot
Slow.01     mondas      19/07/14    4 B       incr       /d/01
Slow.01     mondas      19/07/14    2 KB      incr       /d/backup1
Slow.01     orilla.turbamentis.int 19/07/14  7206 KB   incr       /
Slow.01     orilla.turbamentis.int 19/07/14    4 B       incr       /synology/finance
Slow.01     orilla.turbamentis.int 19/07/14  1077 KB   incr       /synology/pmdg/Docum
ents
Slow.01     orilla.turbamentis.int 20/07/14  181 KB    full      CONSOLE_BACKUP_FILES
```

Figure 3: Using a query and sort order

The specific query to run is specified in the `mminfo` command by using the argument `-q`, followed by a series of either flag¹ or value proposition arguments. In this case, we've used the argument:

`savetime>=24 hours ago`

This introduces the first of the eccentricities of `mminfo`. Whenever you use greater than or less than signs in an `mminfo` command, you should understand that they mean the following:

- Greater than (>) = "More recent than"
- Less than (<) = "Older than"

Another way of considering the process is that `mminfo` converts all time-based arguments to the number of seconds since a January 1, 1970 (GMT). Hence, "savetime>=24 hours ago" means "where the time of the saveset in seconds since January 1, 1970 is greater than the time of 24 hours ago in seconds since January 1, 1970".

The following table provides details of query options in NetWorker's `mminfo` utility:

Attribute	Type	Description
%used	'full' or number	The percentage of the estimated capacity of the volume that has been used. (E.g., "%used>50").
annotation	String	Annotation (description) for an archive saveset.
backup_size	Number	Size of a VBA saveset on either internal VBA storage or a NetWorker device ² .
capacity	String	Estimated capacity of the volume (e.g., "400GB").
checkpoint_id	String	For checkpoint savesets, the ID of the checkpoint.
checkpoint_seq	String	For checkpoint savesets, the sequence of the saveset.

¹ It is best to avoid thinking of the flag options as Boolean options, since `mminfo` does not handle flag negation as the average user would expect.

² Note that NetWorker currently does not limit this field to use for VBA backups only.

Attribute	Type	Description
checkpoint-restart	Flag	Matches savesets with the 'checkpoint restart' enabled option.
client	String	Name of the client associated with the backup.
clientid	String	Client identifier (unique) associated with the backup ³ .
cloneid	Number	The unique identifier of a saveset clone.
clonetime	Time	Date/timestamp that a saveset clone was generated.
clretent	Time	The date/timestamp that a saveset clone will expire.
continued	Flag	Matches savesets that have continued on to or from another volume.
copies	Number	Number of clones of the saveset (this includes the original backup as well, if still present).
cover	Flag	Matches 'cover' savesets.
dsa	Flag	Matches NDMP savesets that have been transferred across to a NetWorker server or storage node and saved to a standard volume.
family	String	The type of media family (e.g., 'disk', 'tape').
first	Number	Offset to the first byte of the saveset within a section. (For multiplexed backups.)
full	Flag	Matches full volumes.
group	String	The group the saveset was generated out of. (Blank if it was a manual backup.)
incomplete	Flag	Matches savesets that did not complete.
inuse	Flag	Matches volumes that are currently in use.
labeled	Time	Date/timestamp that the volume was most recently labelled. (Note the spelling of this option.)
last	Number	Offset of the last byte of the saveset within a section.
level	String	Any valid NetWorker level string. This will be one of: <ul style="list-style-type: none"> o through to 9 full (or 'f'), incr (or 'i'), migration (or 'm').
location	String	Location of the volume.
manual	Flag	Matches volumes that are manually recyclable.
mediafile	Number	For tape-based backups, the file number (measured in EOFs written to tape) where the saveset starts. Always 0 for disk based backups.
mediarec	Number	For tape-based backups, the media record number (internal to a single media file) within a block of data where the saveset instance starts. Always 0 for disk based backups.
mounts	Number	Number of times the label of the volume is read – does not necessarily have a 1:1 correlation with the number of times the volume is mounted into a device.
name	String	Name of a save set.
ndmp	Flag	Matches NDMP savesets performed directly by NDMP hosts.
near	Flag	Matches volumes that are flagged in the media database as 'nearline'.
next	Number	Next media file that will be written.
nfiles	Number	Number of client files in the saveset. This is particularly useful for filesystem based backups.
nrec	Number	Next media record that will be written.

³ It is more correct to say that savesets are tagged as belonging to a particular client ID rather than a particular client. The client ID is mapped by the media database to specific client names; this allows the user to rename a client but keep existing backups for that client.

Attribute	Type	Description
olabel	Time	Date/timestamp that the volume was first (originally) labelled.
pool	String	Pool name.
pssid	Number/String	Saveset ID (short or long format, as per 'ssid') of the first saveset in a saveset series ⁴ .
raw	Flag	Matches raw savesets. (An attribute reserved for particular modules.)
read	Number	Number of KB read from the volume.
readonly	Flag	Matches volumes that are read-only.
recoverable	Flag	Matches savesets whose browse time has expired but have not yet expired.
recycled	Number	Number of times the volume has been recycled.
rehydrated	Flag	Matches those savesets that have been rehydrated from Avamar deduplicated savesets.
rolledin	Flag	Matches savesets that have the 'rolled-in' flag.
savesets	Number	Number of savesets (or partial ones thereof) on a volume.
savetime	Time	Date/timestamp of the backup (from the client clock).
scan	Flag	Matches volumes that have been flagged as requiring scanning.
smartmedia	Flag	Matches volumes that are flagged in the media database as belonging to 'SmartMedia' ⁵ .
snap	Flag	Matches backups flagged as snapshots.
ssaccess	Time	Date/timestamp of when the saveset was last accessed for backup or recovery purposes.
ssbrowse	Time	Date/timestamp that the browse period for the saveset will expire.
ssbundle	String	A save set bundle identifier; this is used in certain instances where multiple savesets are staged out together.
sscomp	Time	Date/timestamp of when the saveset was completed.
sscreate	Time	Date/timestamp of the backup (from the server clock).
ssid	Number/String	Can be specified either as a number up to 10 characters long, or a long form format (53 characters long).
ssinsert	Time	Date/timestamp that the saveset was most recently added to the media database. This is typically either going to be the saveset creation time (via a backup) or when it was scanned in.
ssrecycle	Flag	Matches savesets that are recyclable (browse and retention have expired).
ssretent	Time	Date/timestamp that the retention time for the saveset will expire.
suspect	Flag	Whether or not NetWorker has flagged the saveset as suspect.
synthetic_full	Flag	Matches those savesets that are full, and tagged as synthetic full backups.
totalsize	Number	Total size of the saveset, in bytes.
type	String	Media type (e.g. adv_file, "LTO Ultrium-4").
valid	Flag	Matches valid savesets. (Note that all savesets are currently marked as valid by existing NetWorker servers.)
validcopies	Number	Number of successful copies of a saveset.
vmname	String	Virtual machine name that a save set belongs to.

⁴ The pssid option was typically associated with the NetWorker media database when savesets had to be split on 2000MB boundaries.

⁵ It is unlikely that any existing NetWorker server would still make use of the SmartMedia flag.

Attribute	Type	Description
volaccess	Time	Date/timestamp the volume was last access, regardless of whether it was for reading, writing.
valid	String	Unique volume identifier number.
volrecycle	Flag	Matches volumes that are currently recyclable.
volretent	Time	Matches the longest retention time for any saveset on the volume.
volume	String	The volume name.
written	String	Amount of data written to the volume, in KB.

Note that there are quick-use arguments for mminfo:

- -c *client* – Retrieve entries for named client only
- -N *name* – Retrieve entries for named saveset only
- -t *time* – Retrieve entries for the time specified.

In each instance, these can be used instead of the query specification option listed in the table above. For example, the following two mminfo commands will achieve the same results:

```
# mminfo -q "client=archon"
# mminfo -c archon
```

There is no *correct* method above, but this guide will focus solely on the former.

8.2.1 Specifying Multiple Queries

Multiple query options in mminfo are separated by a comma. The same query argument can be used multiple times, for example, consider the query:

```
# mminfo -q "client=archon,client=skaro,savetime>=24 hours ago"
```

```
[root@orilla ~]# mminfo -q "client=archon,client=skaro,savetime>=24 hours ago"
volume      client      date        size        level  name
Standard.01 archon      19/07/14    3879 MB     incr   /
Standard.01 archon      19/07/14    3052 MB     incr   /
Standard.01 archon      19/07/14    46 KB      incr   /Volumes/Encrypted
Standard.01 archon      19/07/14    11 KB      incr   /Volumes/Encrypted
Standard.01 archon      19/07/14    9940 KB     incr   /Volumes/Storage
Standard.01 archon      19/07/14    13 KB      incr   /Volumes/Storage
Standard.01 skaro       19/07/14    1168 MB     incr   C:\
Standard.01 skaro       19/07/14    113 MB     incr   C:\
Standard.01 skaro       19/07/14    20 KB      incr   DISASTER_RECOVERY:\
Standard.01 skaro       19/07/14    20 KB      incr   DISASTER_RECOVERY:\
Standard.01 skaro       19/07/14    9303 KB     full   WINDOWS ROLES AND FEATURES:\
Standard.01 skaro       19/07/14    9303 KB     full   WINDOWS ROLES AND FEATURES:\
Standard.01 skaro       19/07/14    30 MB      full   \\?\GLOBALROOT\Device\Harddi
skVolume2\
Standard.01 skaro       19/07/14    30 MB      full   \\?\GLOBALROOT\Device\Harddi
skVolume2\
```

Figure 4: Querying on multiple clients

In this scenario, the query is:

Retrieve all backups for (the client archon or the client skaro) generated in the last 24 hours.

Consider a slightly different query, more focused on time:

```
# mminfo -q "client=archon,savetime>=3 weeks ago,savetime<=2 weeks ago"
```

This yields:

```
[root@orilla ~]# mminfo -q "client=archon,savetime>=3 weeks ago,savetime<=2 weeks ago"
volume      client      date        size        level name
Standard.01 archon      29/06/14    8549 MB     incr /
Standard.01 archon      30/06/14    7440 MB     incr /
Standard.01 archon      02/07/14    10 GB       incr /
Standard.01 archon      03/07/14    2993 MB     incr /
Standard.01 archon      05/07/14    3900 MB     incr /
Standard.01 archon      02/07/14    47 MB       full /Volumes/Encrypted
Standard.01 archon      03/07/14    237 KB      incr /Volumes/Encrypted
Standard.01 archon      06/07/14    84 MB       incr /Volumes/Encrypted
Standard.01 archon      29/06/14    48 KB       incr /Volumes/Storage
Standard.01 archon      30/06/14    4846 MB     incr /Volumes/Storage
Standard.01 archon      05/07/14    105 GB      incr /Volumes/Storage
```

Figure 5: Query based on a date range

In this scenario, the query is:

Retrieve all backups for the client archon generated between (3 and 2 weeks ago).

Consider the difference between this query and the previous query. In the previous query, using the same field twice resulted in an ‘or’ operation – all backups for the client archon *or* the client skaro. In this query, using the same field resulted in an ‘and’ operation – all backups that are more recent than 3 weeks ago *and* less recent than 2 weeks ago.

This is perhaps one of the areas where beginners in mminfo get most confused, and it’s important to understand that mminfo queries are *not* some form of SQL. Since the arguments and flags that might be queried on are limited to a specific function, mminfo will intelligently interpret the arguments provided. So:

- *client=archon,client=skaro* logically can only mean one thing – where the client is ‘archon’ or the client is ‘skaro’.
- *savetime>=3 weeks ago,savetime<=2 weeks ago* can also logically only mean one thing – where the backup time occurred within the last 3 weeks, and where the backup time occurred no recently than 2 weeks ago.

In both cases, the query with an alternate logical operand would make no sense. After all, a saveset can never be one that belongs to both the client *archon* and the client *skaro*⁶. Equally, if the savetime query were based on *or* rather than *and*, it would return *all* savesets – the first would match every saveset done in the last three weeks, and the second would match every saveset done more than 2 weeks ago.

8.2.2 Many query options

It’s entirely normal to use mminfo to narrow down the selection to a highly specific type of saveset. For instance, consider the scenario where you wanted to find all *full* backups of the special saveset ‘WINDOWS ROLES AND FEATURES:’ for the client skaro performed between 2 and 4 weeks ago. This would resemble the following:

⁶ The saveset *instance*. The same saveset *name* can be shared between two clients, but this is no different from knowing two different people called Isobel.


```
# mminfo -q 'client=skaro,savetime>=4 weeks ago,savetime<=2 weeks ago,level=full,name=WINDOWS ROLES AND FEATURES:\'
```

```
[root@orilla ~]# mminfo -q 'client=skaro,savetime>=4 weeks ago,savetime<=2 weeks ago,level=full,name=WINDOWS ROLES AND FEATURES:\'
```

volume	client	date	size	level	name
Standard.01	skaro	23/06/14	9295 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	24/06/14	9295 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	25/06/14	9294 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	26/06/14	9295 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	27/06/14	9294 KB	full	WINDOWS ROLES AND FEATURES:\

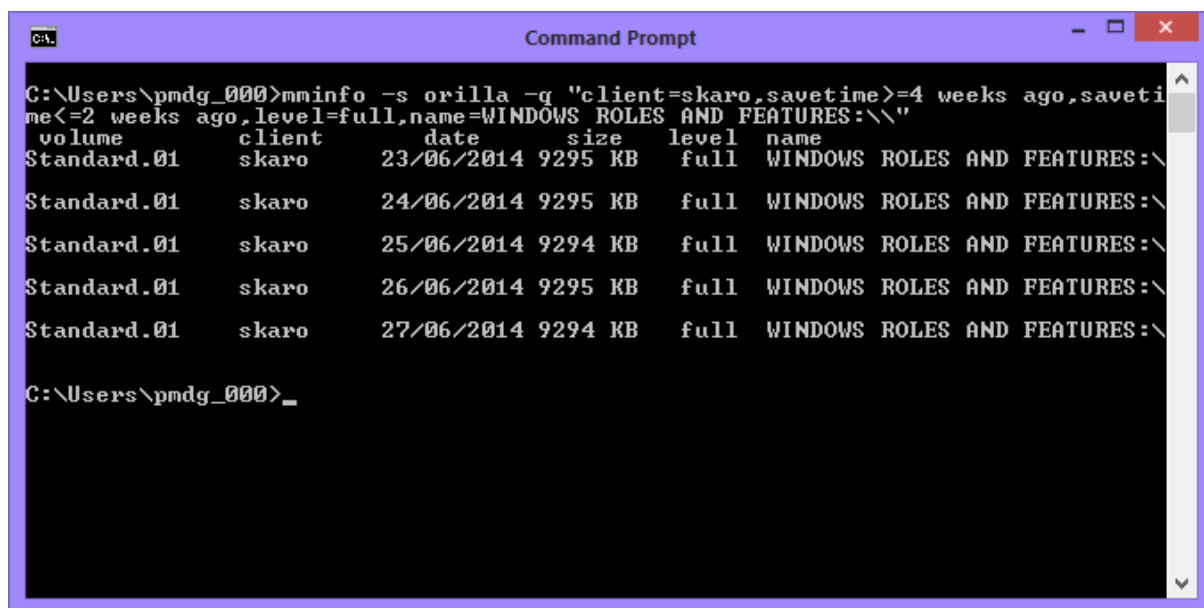
Figure 6: Narrowing down a query

While mminfo is almost entirely platform neutral in its query construction, the use of backslash in Windows saveset names is an exception. The above query, using *single* quotes instead of double quotes, is a Unix-only query.

An alternate query that will work on both Windows *and* Unix systems would be:

```
# mminfo -q "client=skaro,savetime>=4 weeks ago,savetime<=2 weeks ago,level=full,name=WINDOWS ROLES AND FEATURES:\\"
```

In this scenario, we're 'escaping' the backslash in the Windows saveset with a double-backslash. This query works on both platforms – for instance:



```
C:\Users\pmdg_000>mminfo -s orilla -q "client=skaro,savetime>=4 weeks ago,savetime<=2 weeks ago,level=full,name=WINDOWS ROLES AND FEATURES:\\"
```

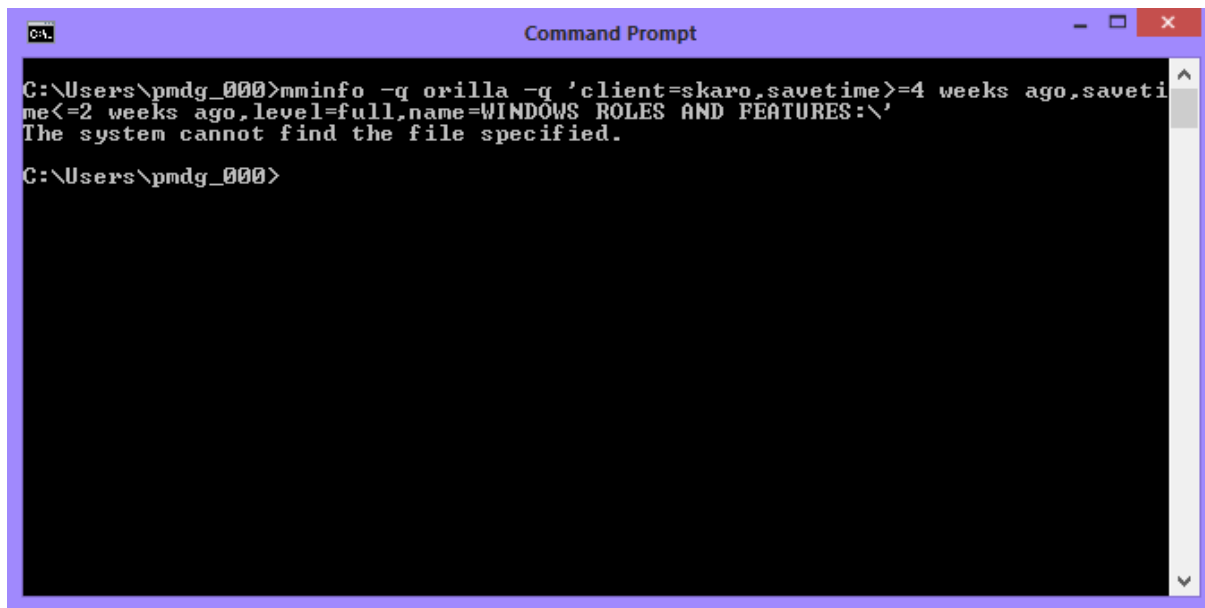
volume	client	date	size	level	name
Standard.01	skaro	23/06/2014	9295 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	24/06/2014	9295 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	25/06/2014	9294 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	26/06/2014	9295 KB	full	WINDOWS ROLES AND FEATURES:\
Standard.01	skaro	27/06/2014	9294 KB	full	WINDOWS ROLES AND FEATURES:\

```
C:\Users\pmdg_000>_
```

Figure 7: mminfo query run from Windows

In this instance, the query isn't being run from the NetWorker server itself – instead, it's being run from the client skaro; as such, mminfo must be supplied with another argument – the server name ("-s orilla").

In case you're wondering, running the query with single quotes on Windows will result in a somewhat odd result:



```
C:\Users\pmdg_000>mminfo -q orilla -q 'client=skaro,savetime>=4 weeks ago,savetime<=2 weeks ago,level=full,name=WINDOWS ROLES AND FEATURES:\'
The system cannot find the file specified.

C:\Users\pmdg_000>
```

Figure 8: Example of mminfo queries using single quotes on Windows

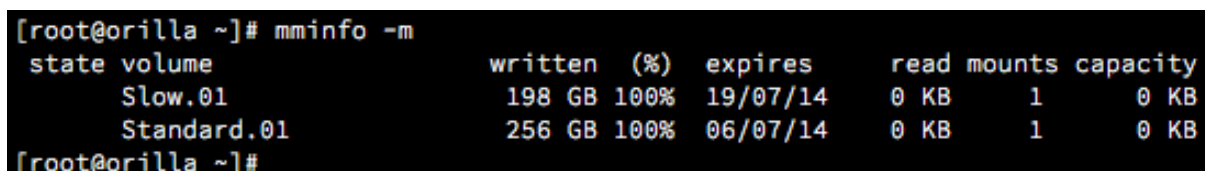
This leads us to some important rules to follow when scripting mminfo queries so that they may be reliably run on both Windows and Unix/Linux platform as required:

1. Always enclose the query parameters in double quotes, not single quotes.
2. Use the '\\' convention of escaping a backslash for Windows paths.

8.3 Media Queries

So far we've concentrated on *saveset* queries – where the focus of the output has been on individual backups. The mminfo utility however has an alternate mode that focuses on *media* details – volumes.

The base media report is accessed via 'mminfo -m', and the output resembles the following:



```
[root@orilla ~]# mminfo -m
state volume                written  (%)  expires    read mounts capacity
      Slow.01                198 GB 100%  19/07/14   0 KB   1     0 KB
      Standard.01            256 GB 100%  06/07/14   0 KB   1     0 KB
[root@orilla ~]#
```

Figure 9: mminfo -m output (disk volumes)

The above output is for a server running only with disk backup devices. Tape based systems will obviously present more volumes:


```
[root@tara ~]# mminfo -m
state volume                written (%) expires    read mounts capacity
      800840L4              2960 MB full  01/06/15    0 KB      2   2000 MB
      800841L4              2707 MB full  01/06/15    0 KB      2   2000 MB
      800842L4              2706 MB full    undef     0 KB      2   2000 MB
      800843L4              2706 MB full    undef     0 KB      2   2000 MB
      800844L4              2707 MB full    undef     0 KB      2   2000 MB
      800845L4              2706 MB full    undef     0 KB      2   2000 MB
      800846L4              2706 MB full    undef     0 KB      2   2000 MB
      800847L4              2707 MB full    undef     0 KB      2   2000 MB
      800848L4              2706 MB full    undef     0 KB      2   2000 MB
      800849L4              2706 MB full    undef     0 KB      2   2000 MB
      800850L4              2707 MB full    undef     0 KB      2   2000 MB
      800851L4              2706 MB full    undef     0 KB      2   2000 MB
      800852L4              2707 MB full    undef     0 KB      2   2000 MB
      800853L4              2706 MB full    undef     0 KB      2   2000 MB
```

Figure 10: mminfo -m output (tape volumes)

Additionally, the media report offers a *verbose* mode that outputs three additional fields – the volume ID, the next file marker, and the volume type. (The file marker is not referenced in disk backup devices such as `adv_file` and Data Domain Boost. It refers to tapes – physical or virtual.) Output from the verbose media report resembles the following:

```
[root@tara ~]# mminfo -mv
state volume                written (%) expires    read mounts capacity valid    next type
      800840L4              2960 MB full  01/06/15    0 KB      2   2000 MB 4287266918 3 LTO Ultrium-4
      800841L4              2707 MB full  01/06/15    0 KB      2   2000 MB 9076838 4 LTO Ultrium-4
      800842L4              2706 MB full    undef     0 KB      2   2000 MB 4270489702 3 LTO Ultrium-4
      800843L4              2706 MB full    undef     0 KB      2   2000 MB 4253712511 3 LTO Ultrium-4
      800844L4              2707 MB full    undef     0 KB      2   2000 MB 4220158079 3 LTO Ultrium-4
      800845L4              2706 MB full    undef     0 KB      2   2000 MB 4236935295 3 LTO Ultrium-4
      800846L4              2706 MB full    undef     0 KB      2   2000 MB 4169826458 3 LTO Ultrium-4
      800847L4              2707 MB full    undef     0 KB      2   2000 MB 4203380890 3 LTO Ultrium-4
      800848L4              2706 MB full    undef     0 KB      2   2000 MB 4186603674 3 LTO Ultrium-4
      800849L4              2706 MB full    undef     0 KB      2   2000 MB 4153049267 3 LTO Ultrium-4
```

Figure 11: mminfo -mv (verbose media report) output

8.4 Specifying the Report Columns

Once we've sorted out how to assemble mminfo queries, the next step is to select the output that we want to see – this is achieved via the *report* (-r) argument.

To start with, consider the scenario where all we want to see is the name, size, date and level of every backup done for the client 'mondas' in the last two weeks, ordered by time, with the most recent backups first. The query for this would resemble the following:

```
# mminfo -q "client=mondas,savetime>=2 weeks ago" -r
savetime,name,sumsize,level -ot
```

```
[root@orilla ~]# mminfo -q "client=mondas,savetime>=2 weeks ago" -r savetime,name,
sumsize,level -otR
  date   name                size  lvl
20/07/14 /d/backup1          4 B incr
20/07/14 /boot                4 B incr
20/07/14 /d/01                4 B incr
20/07/14 /                    12 MB incr
19/07/14 /d/backup1          2 KB incr
19/07/14 /boot                4 B incr
19/07/14 /d/01                4 B incr
19/07/14 /                    9251 KB incr
18/07/14 /d/backup1          2 KB incr
18/07/14 /boot                4 B incr
18/07/14 /d/01                4 B incr
18/07/14 /                    9128 KB incr
17/07/14 /d/backup1          4 B incr
17/07/14 /boot                4 B incr
17/07/14 /d/01                4 B incr
```

Figure 12: Specifying report columns in mminfo

The report argument allows us to not only specify the columns that we wish to see, but also the width of the columns. For instance, the *savetime* output is actually a date/timestamp, but the default output width results in only showing the date. If we wanted to see the time as well, we could run the command:

```
# mminfo -q "client=mondas,savetime>=2 weeks ago" -r
"savetime(18),name,sumsize,level" -ot
```

With the revised command in place, the output will resemble the following:

```
[root@orilla ~]# mminfo -q "client=mondas,savetime>=2 weeks ago" -r "savetime(18
),name,sumsize,level" -otR
  date   time   name                size  lvl
20/07/14 21:02:23 /d/backup1          4 B incr
20/07/14 21:02:18 /boot                4 B incr
20/07/14 21:02:13 /d/01                4 B incr
20/07/14 21:00:38 /                    12 MB incr
19/07/14 21:01:43 /d/backup1          2 KB incr
19/07/14 21:01:38 /boot                4 B incr
19/07/14 21:01:33 /d/01                4 B incr
19/07/14 21:00:38 /                    9251 KB incr
18/07/14 21:01:47 /d/backup1          2 KB incr
18/07/14 21:01:42 /boot                4 B incr
```

Figure 13: Specifying column width in report output

The width option can be used to change several ‘conventional’ details. For instance:

- Using a width of 10 or more against a size field that normally outputs in ‘X yB’ (e.g., “2088 MB”) will result in the size field being output in bytes;
- Using a width of 53 or more against the saveset ID yields the *long form* saveset ID, which can in turn be matched against specific files on advanced file type devices.

For example, consider the following query:

```
# mminfo -q "savetime>=24 hours ago" -r "name,ssid,ssid(53)"
```

The output from this command might resemble the following:

```
[root@orilla ~]# mminfo -q "savetime>=24 hours ago" -r "name,ssid,ssid(53)"
name          ssid          ssid
/             2949360000    b56017d4-00000006-afcba980-53cba980-00
52d93c-b6f40c2b
/Volumes/Encrypted 2815144314    c85e3a67-00000006-a7cbb17a-53cbb17a-00
5ad93c-b6f40c2b
/Volumes/Storage  2831921401    868638cd-00000006-a8cbb0f9-53cbb0f9-00
59d93c-b6f40c2b
/             2781592018    ae3fcf45-00000006-a5cbb9d2-53cbb9d2-00
5cd93c-b6f40c2b
/Volumes/Alteran/Documents/Music 2731264574    2a231bf3-00000006-a2cbca3e-53cbca3e-
005fd93c-b6f40c2b
```

Figure 14: Getting the long-form saveset ID

To demonstrate the use of the long-form saveset ID to associate with an actual file, the command above was run on a server with `adv_file` devices. When a correlating ‘find’ command was run against the disk backup filesystem, the file containing the saveset was found:

```
[root@orilla ~]# mminfo -q "savetime>=24 hours ago" -r "name,ssid,ssid(53)"
name          ssid          ssid
/             2949360000    b56017d4-00000006-afcba980-53cba980-00
52d93c-b6f40c2b
/Volumes/Encrypted 2815144314    c85e3a67-00000006-a7cbb17a-53cbb17a-00
5ad93c-b6f40c2b
/Volumes/Storage  2831921401    868638cd-00000006-a8cbb0f9-53cbb0f9-00
59d93c-b6f40c2b
/             2781592018    ae3fcf45-00000006-a5cbb9d2-53cbb9d2-00
5cd93c-b6f40c2b
/Volumes/Alteran/Documents/Music 2731264574    2a231bf3-00000006-a2cbca3e-53cbca3e-
005fd93c-b6f40c2b
```

Figure 15: Finding files on `adv_file` devices based on the long-form saveset ID

A large number of the potential report options have already been listed in Table 1 for the query options, with the key consideration that query *flag* options are not valid as report options.

Additional report options are outlined in the table below:

Attribute	Description
attrs	Extended saveset attributes. This provides additional information about savesets.
avail	Volume availability – i.e., <i>where</i> the volume is. For the time being, this is limited to: <ul style="list-style-type: none"> n – Nearline (within a jukebox) ov – SmartMedia⁷ managed volume
clflags	Clone flags for this specific saveset clone instance. Can be: <ul style="list-style-type: none"> a – Aborted i – Incomplete s – Suspect E – Eligible for recycling
fragflags	Saveset fragment summary flags, in the same format as ‘sumflags’.
fragsize	The calculated size of a saveset fragment. If specified without width, this will output in conventional format – e.g., “2975 MB”. If a width of 10 or larger is specified, it will output in bytes.
last	Offset of the last byte of the saveset in the current fragment.

⁷ Given SmartMedia has been end-of-life for some time, it is likely the only flag that will be observed here is ‘n’ for nearline media.

Attribute	Description
newline	Force a newline into the output. Use “newline(<i>n</i>)” to generate a particular number of carriage returns into the output.
nsavetime	Save time, expressed as the number of seconds since January 1, 1970 (GMT).
savesets	Number of savesets on a volume.
space	Space column. Use “space(<i>n</i>)” to generate a particular number of spaces.
ssflags	Standard saveset flags. This will include one or more of the following (depending on the state or type of the saveset): <ul style="list-style-type: none"> • C – A continued saveset • v – A valid saveset • r – A recoverable saveset (“purged”) • E – A saveset that is eligible for recycling • N – An NDMP generated saveset • i – An incomplete saveset • R – A raw saveset • P – A snapshot saveset • K – A cover saveset • I – An in-progress saveset • F – A finished saveset • k – A checkpoint-restart enabled saveset
state	Volume state information (blank if none set): <ul style="list-style-type: none"> • E – eligible for recycling • M – a volume flagged for manual recycling • X – a volume flagged for manual recycling that <i>is</i> eligible to be recycled • A – an archive or migration volume
sumflags	Volume saveset summary flags. This consists of two types of summary information – the part of the saveset on the volume and the saveset status. For the part of the saveset on the volume: <ul style="list-style-type: none"> • c – The complete saveset is on this volume • h – The head (start) of the saveset is on this volume • m – A middle part of the saveset is on this volume • t – The tail (end) of the saveset is on this volume For the saveset status: <ul style="list-style-type: none"> • b – The saveset is browseable (i.e., still in client indices) • r – The saveset is recoverable (i.e., not in client indices) • a – The saveset was aborted • i – The saveset is currently in-progress
sumsize	Size of all the sections of the saveset <i>on this volume</i> . If specified without width, this will output in conventional format – e.g., “2975 MB”. If a width of 10 or larger is specified, it will output in bytes.
volattrs	Extended volume attributes.
volflags	Potential volume flags (blank if none set): <ul style="list-style-type: none"> • d – dirty (volume is currently in use for writing) • r – read only • S – scan needed

Table 2: Additional report fields

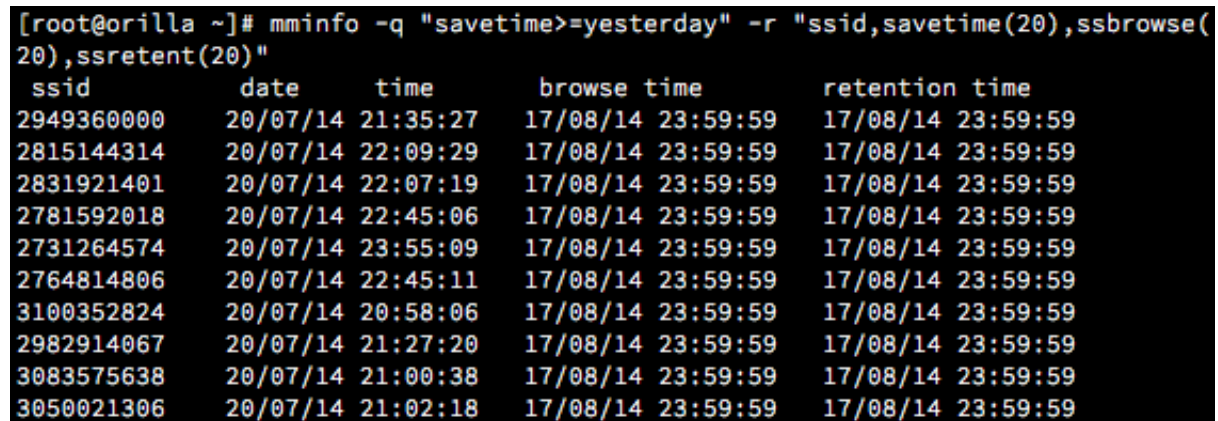
Just with query parameters, the report specification for an mminfo command can become quite complex. For instance, the *default* output fields are:

```
-r volume,client,savetime,sumsize,level,name
```

The report fields that you might choose to use will be largely governed by the information that you're looking for. For instance, to print a list of all saveset IDs generated since yesterday, including their full date/timestamp for savetime, browse expiration date and retention expiration date, you would use the command:

```
# mminfo -q "savetime>=yesterday" -r  
"ssid,savetime(20),ssbrowse(20),ssretent(20)"
```

This would yield output resembling the following:



```
[root@orilla ~]# mminfo -q "savetime>=yesterday" -r "ssid,savetime(20),ssbrowse(20),ssretent(20)"  
ssid      date      time      browse time      retention time  
2949360000 20/07/14 21:35:27 17/08/14 23:59:59 17/08/14 23:59:59  
2815144314 20/07/14 22:09:29 17/08/14 23:59:59 17/08/14 23:59:59  
2831921401 20/07/14 22:07:19 17/08/14 23:59:59 17/08/14 23:59:59  
2781592018 20/07/14 22:45:06 17/08/14 23:59:59 17/08/14 23:59:59  
2731264574 20/07/14 23:55:09 17/08/14 23:59:59 17/08/14 23:59:59  
2764814806 20/07/14 22:45:11 17/08/14 23:59:59 17/08/14 23:59:59  
3100352824 20/07/14 20:58:06 17/08/14 23:59:59 17/08/14 23:59:59  
2982914067 20/07/14 21:27:20 17/08/14 23:59:59 17/08/14 23:59:59  
3083575638 20/07/14 21:00:38 17/08/14 23:59:59 17/08/14 23:59:59  
3050021306 20/07/14 21:02:18 17/08/14 23:59:59 17/08/14 23:59:59
```

Figure 16: Using multiple width fields in a custom report

As you would expect, the report options can equally be used to determine information about volume usage. For example, consider a scenario where you want to report on the following for each volume:

- The volume type
- The volume name
- When the volume was first labelled
- When the volume was most recently labelled
- How many times the volume has been recycled

This information might be generated periodically for physical tapes in order to enact an aging policy.

The command used would resemble the following:

```
# mminfo -q "olabel>10 years ago" -r  
type,volume,olabel,labeled,recycled
```

Note that the query used here is relatively arbitrary. By default, even if the fields being output are based on volume information, NetWorker will not output details for empty volumes. As such, we *force* their inclusion by specifying a volume based query based on a timeframe we know will include *all* volumes. In this case, 'olabel>10 years ago' will select all volumes that were originally labelled in the last 10 years – for a temporary lab server, this is more than sufficient. Based on your environment, the command may need to be varied a little.


```
[root@tara ~]# mminfo -q "olabel>10 years ago" -r type,volume,olabel,labeled,recycled
type volume orig lbl labeled rcyc
LTO Ultrium-4 800840L4 10/07/14 18/07/14 1
LTO Ultrium-4 800841L4 10/07/14 18/07/14 1
LTO Ultrium-4 800842L4 10/07/14 10/07/14 0
LTO Ultrium-4 800843L4 10/07/14 10/07/14 0
LTO Ultrium-4 800844L4 10/07/14 10/07/14 0
LTO Ultrium-4 800845L4 10/07/14 10/07/14 0
LTO Ultrium-4 800846L4 10/07/14 10/07/14 0
LTO Ultrium-4 800847L4 10/07/14 10/07/14 0
LTO Ultrium-4 800848L4 10/07/14 10/07/14 0
LTO Ultrium-4 800849L4 10/07/14 10/07/14 0
LTO Ultrium-4 800850L4 10/07/14 10/07/14 0
```

Figure 17: Producing a volume aging report

8.5 Enhancements for VBA Backups

NetWorker's new VBA approach to VMware backups has proven extremely popular in a relatively short period of time, but it does introduce some changes for reporting in *mminfo*. In particular, the default reporting for saveset *names* for VBA backups doesn't include the sort of information that allows the virtual machine to be relatively identified.

For instance:

```
[root@centaur ~]# mminfo -q "pool=Squeeze,savetime>=24 hours ago"
volume client date size level name
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 50 GB full vm:50290268-c12a-f665-507d-0a2a4fa5bd17:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 146 GB full vm:50294e6c-c3cc-07aa-71b1-1f7e9fda7a01:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 20 GB full vm:50299bf6-0828-794e-4e35-e347d20003d7:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 50 GB full vm:5029ab21-320e-1bb5-b7fa-2336c8512db6:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 41 GB full vm:5029ddcb-06c6-2e4d-b881-8ffdf7f09cd47:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 16 GB full vm:5029ef6f-18f1-a3f9-b99c-75f309b13fbc:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 31 GB full vm:5029f29e-3084-f9f5-22cc-f9a7103cc946:c128.turbamentis.int
Squeeze.002 vba-ebr.turbamentis.int 06/04/15 73 GB full vm:5029f49f-8984-3fee-16af-1ad45ab9d8f3:c128.turbamentis.int
Squeeze.003 vba-ebr.turbamentis.int 06/04/15 83 GB full vm:50294827-d10a-793a-4903-ed8570967113:c128.turbamentis.int
Squeeze.003 vba-ebr.turbamentis.int 06/04/15 41 GB full vm:5029dfe4-4476-a54c-48e2-85a39b167a50:c128.turbamentis.int
```

Figure 18: mminfo output featuring VBA virtual machine backups

As can be seen by the above screenshot, a saveset name of *vm:<vm_id>:vcenterName* isn't the best readable format for determining what virtual machines have been backed up, and the other pertinent details of those backups. This can however be resolved with two options in *mminfo*.

The first option is to generate a VBA specific backup report, via the option *-k*. In the most basic format, this output resembles the following:

```
[root@centaur ~]# mminfo -k
```

volume	type	vm_name	date	time	size	ssid	fl	backup_size
Squeeze.002	Data Domain	dbase3	06/04/15	10:32:41	50 GB	2216809558	cr	50 GB
Squeeze.002	Data Domain	nodns-nsn	06/04/15	10:33:21	146 GB	2149700732	cr	146 GB
Squeeze.002	Data Domain	sirius	06/04/15	10:33:11	20 GB	2166477939	cr	20 GB
Squeeze.002	Data Domain	dbase1	06/04/15	10:32:51	50 GB	2200032350	cr	50 GB
Squeeze.002	Data Domain	nodns-nmc	06/04/15	10:33:32	41 GB	2132923526	cr	41 GB
Squeeze.002	Data Domain	nodns-cln	06/04/15	10:36:13	16 GB	2065814823	cr	16 GB
Squeeze.002	Data Domain	win01	06/04/15	10:33:01	31 GB	2183255144	cr	31 GB
Squeeze.002	Data Domain	dbase2	06/04/15	10:34:48	73 GB	2082591954	cr	73 GB
Squeeze.003	Data Domain	win02	06/04/15	10:33:42	83 GB	2116146323	cr	83 GB
Squeeze.003	Data Domain	nodns-nsr	06/04/15	10:33:53	41 GB	2099369115	cr	41 GB

Figure 19: mminfo's new VBA specific report output

The alternate is to reference specific mminfo report fields designed for VBA backups:

- `vmname` – The actual name of the virtual machine
- `backup_size` – The size of the virtual machine backup⁸

For instance, if we wanted to see the virtual machine backups written to the *Squeeze* pool in the last 24 hours, a command such as the following could be used:

```
# mminfo -q "pool=Squeeze,savetime>=24 hours ago" -r
volume,vmname,sumsize,backup_size,level
```

Output for this might resemble the following:

```
[root@centaur ~]# mminfo -q "pool=Squeeze,savetime>=24 hours ago" -r volume,vmname,sumsize,backup_size,level
```

volume	vm_name	size	backup_size	lvl
Squeeze.002	dbase3	50 GB	50 GB	full
Squeeze.002	nodns-nsn	146 GB	146 GB	full
Squeeze.002	sirius	20 GB	20 GB	full
Squeeze.002	dbase1	50 GB	50 GB	full
Squeeze.002	nodns-nmc	41 GB	41 GB	full
Squeeze.002	nodns-cln	16 GB	16 GB	full
Squeeze.002	win01	31 GB	31 GB	full
Squeeze.002	dbase2	73 GB	73 GB	full
Squeeze.003	win02	83 GB	83 GB	full
Squeeze.003	nodns-nsr	41 GB	41 GB	full

Figure 20: Reporting using the 'vmname' and 'backup_size' options

Note that if you generate a query that includes both regular backups and VBA backups, the `vmname` and `backup_size` output options will be blank for conventional backups.

8.6 XM your L

The volume usage output is useful, but in the default output format of mminfo, it doesn't necessarily lend itself to automated report collation where the results are to be *easily* parsed. This leads us to mminfo's output format options. This can be in one of two variants:

- `-xml` – XML or
- `-xcS` – Arbitrary field separation by the string denoted by S.

By rights, the `-xcS` format is intended for use with a single character, but there's no real limit on the number of characters used. For instance, the following is a valid command:

⁸ This will only differ from the conventional size settings when the virtual machine has been backed up to the internal data store on the VBA.

```
# mminfo -q "olabel>10 years ago" -r  
type,volume,olabel,labeled,recycled -xc" gibber jabber "
```

```
[root@tara ~]# mminfo -q "olabel>10 years ago" -r type,volume,olabel,labeled,recycled -xc" gibber jabber "  
type gibber jabber volume gibber jabber orig-label gibber jabber labeled gibber jabber recycled  
LT0 Ultrium-4 gibber jabber 800840L4 gibber jabber 10/07/14 gibber jabber 18/07/14 gibber jabber 1  
LT0 Ultrium-4 gibber jabber 800841L4 gibber jabber 10/07/14 gibber jabber 18/07/14 gibber jabber 1  
LT0 Ultrium-4 gibber jabber 800842L4 gibber jabber 10/07/14 gibber jabber 10/07/14 gibber jabber 0  
LT0 Ultrium-4 gibber jabber 800843L4 gibber jabber 10/07/14 gibber jabber 10/07/14 gibber jabber 0
```

Figure 21: Arbitrary field separation in mminfo reports

Unless you're an absolute die-hard fan of certain comedic legal television shows based in Boston, it's unlikely you'd want to fill your reports with gibber jabber, so a more likely field separator would be the comma:

```
# mminfo -q "olabel>10 years ago" -r  
type,volume,olabel,labeled,recycled -xc,
```

```
[root@tara ~]# mminfo -q "olabel>10 years ago" -r type,volume,olabel,labeled,recycled -xc,  
type,volume,orig-label,labeled,recycled  
LT0 Ultrium-4,800840L4,10/07/14,18/07/14,1  
LT0 Ultrium-4,800841L4,10/07/14,18/07/14,1  
LT0 Ultrium-4,800842L4,10/07/14,10/07/14,0  
LT0 Ultrium-4,800843L4,10/07/14,10/07/14,0  
LT0 Ultrium-4,800844L4,10/07/14,10/07/14,0  
LT0 Ultrium-4,800845L4,10/07/14,10/07/14,0  
LT0 Ultrium-4,800846L4,10/07/14,10/07/14,0  
LT0 Ultrium-4,800847L4,10/07/14,10/07/14,0  
LT0 Ultrium-4,800848L4,10/07/14,10/07/14,0
```

Figure 22: Producing comma-separated output from mminfo

Comma separated output is often sufficient for mminfo output in order to import the results into spreadsheets. However, comma separated output by itself may not render output that can be reliably interpreted (e.g., it is conceivable for a variety of fields such as the saveset name and group name to include a comma).

The most reliable way of producing machine results that may be parsed by software is by producing output in XML format, which resembles the following:


```
[root@tara ~]# mminfo -q "olabel>10 years ago" -r type,volume,olabel,labeled,recycled -xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mminfo-tabular-report [
<!ELEMENT result (field*)>
<!ELEMENT field (annotation | attributes | avail | backup_size | barcode | browse-time | capacity | checkpoint-id | checkpoint-restart | checkpoint-seq | client | client-ID | clone-flags | clone-id | clone-retent | clone-time | continued | copies | cover | date-time | dedupe-saveset | dsa | expires | family | files-saved | first | flags | frag-flags | frag-size | full | group | in-use | incomplete | labeled | last | level | location | manual | media-file | media-mark | metric | mounts | name | ndmp | near | newline | next | next-rec | orig-label | percent-used | pool | prev-ssid | raw | rdz | read | read-only | record | recoverable | recycled | rehydrated-saveset | retention-time | rolled-in | savesets | savetime | scan | smartmedia | snap | space | ss-access | ss-completed | ss-created | ss-expires | ss-inserted | ss-recycle | ssbundle | ssflags | ssid | sum-size | suspect | syntheticfull-saveset | total | type | valid | validcopies | vm_name | vol-access | vol-attributes | vol-recycle | volflags | volid | volstate | volume | written)>
<!ELEMENT annotation (string-field)>
<!ELEMENT attributes (string-field)>
<!ELEMENT avail (boolean-field)>
<!ELEMENT backup_size (bytes-field)>
<!ELEMENT barcode (string-field)>
<!ELEMENT browse-time (time-field)>
<!ELEMENT capacity (kilobytes-field)>
```

Figure 23: XML mminfo output

You might note from the above output that there are not yet any actual report details included. Being XML, the output includes all the metadata required to parse the report. The actual report itself follows the metadata, and will resemble the following:

```
<!ELEMENT time-field (#PCDATA)>
]>

<mminfo-tabular-report>
<result>
<type>LT0 Ultrium-4</type>
<volume>800840L4</volume>
<orig-label>10/07/14</orig-label>
<labeled>18/07/14</labeled>
<recycled>1</recycled>
</result>
<result>
<type>LT0 Ultrium-4</type>
<volume>800841L4</volume>
<orig-label>10/07/14</orig-label>
<labeled>18/07/14</labeled>
<recycled>1</recycled>
</result>
<result>
<type>LT0 Ultrium-4</type>
<volume>800842L4</volume>
<orig-label>10/07/14</orig-label>
<labeled>10/07/14</labeled>
<recycled>0</recycled>
</result>
```

Figure 24: Actual XML data content in mminfo XML output

We will cover an example of processing mminfo XML output in the scripting section that follows.

8.7 Stepping it up with a scripting language

8.7.1 Example: Script to determine order of media to scan

NetWorker does not, in itself, contain a scripting language as such. That being said, the CLI is robust and readily accessible with any reasonable scripting language, and the output of mminfo feeds into many scripts around the world.

For one example of scripting, consider the *mminfo -V* command. This gives verbose details about savesets, and is particularly useful in tape or virtual tape environments where savesets span multiple volumes. For instance:

```
[root@tara ~]# mminfo -q "ssid=2680958137" -V
volume          client          size level name
ssid           save time    date      time      browse clretent
first         last file  rec valid      total fl
800843L4        tara.pmdg.lab  2052 MB manual /root
2680958137  1405889692    21/07/14 06:54:52 21/08/14 21/07/15
0 2101914319    2    0 4189978912 156082195864 hb
800844L4        tara.pmdg.lab  2695 MB manual /root
2680958137  1405889692    21/07/14 06:54:52 21/08/14 21/07/15
2101914320 4861776151    2    1 4156424480 156082195864 mb
800846L4        tara.pmdg.lab  2695 MB manual /root
2680958137  1405889692    21/07/14 06:54:52 21/08/14 21/07/15
7622421368 10382544327    2    1 4122870071 156082195864 mb
800847L4        tara.pmdg.lab  2695 MB manual /root
2680958137  1405889692    21/07/14 06:54:52 21/08/14 21/07/15
10382544328 13142667287    2    1 4106092855 156082195864 mb
800848L4        tara.pmdg.lab  2695 MB manual /root
2680958137  1405889692    21/07/14 06:54:52 21/08/14 21/07/15
4861776152 7622421367    2    1 4139647287 156082195864 mb
800849L4        tara.pmdg.lab  2695 MB manual /root
2680958137  1405889692    21/07/14 06:54:52 21/08/14 21/07/15
13142667288 15903312503    2    1 4089315665 156082195864 mb
```

Figure 25: Very verbose mminfo output

When a saveset that is no longer browseable needs to be scanned back in, it is typically recommended to rescan it in the correct volume order, from start to finish. If a saveset only exists on one volume, that's simple. If it exists on two or even three, then the *fragflags* can be used to identify the head, tail and middle segment of the saveset.

You should read the man page/help documentation for the full detail of mminfo's -V option; however, each section of output is broken into to three lines, with the first field on the first line being the volume name, and the last field on the third line being the saveset fragment flags (fragflags).

The first two entries on the third line contain the information critical for determining the order in which a saveset should be scanned in – these are the offsets of the first and last bytes of the saveset contained within *this fragment* of the saveset.

As per the example above, if a saveset spans a larger number of volumes, the output order of mminfo -V may not in fact be the order in which the saveset was generated. To correctly determine this, it's necessary to run a more conventional mminfo command and assemble the order based on the starting and ending position of each fragment within the total saveset size.

In order to correctly interpret the order in which a saveset should be (re)scanned in, the volumes need to be aligned based on the first and last offset bytes of each saveset fragment reported.

The command *mminfo -V* does not lend itself well to scripting. However, we only require a few specific details, and these can be determined using the report options *volume*, *first*, and *last*. A complete Perl script (named 'volume-order.pl') to determine the order is set out below:

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

#####
# Modules
```

```
#####
use strict;
use Getopt::Std;
use File::Basename;
use Math::BigInt lib => 'Calc';

#####
# Global variables
#####
my %opts = ();
my $self = basename($0);
my $ssid = -1;
my $pool = "(undef)";
my %fragments = ();
my $verbose = 0;

#####
# Functions
#####

sub usage {
    print <<EOF;
$self [-h] -S ssid [-b pool] [-v]

Where:

    -h      Print this help and exit.
    -S ssid  Print information for saveset denoted by
ssid.
    -b pool  Restrict search to nominated pool.
    -v      Enable verbose mode.

For use with savesets that span multiple tapes, this command
prints out the order in which savesets appear on the tape(s)
for correct rescanning.
EOF

    if (@_+0 != 0) {
        my @messages = @_;
        foreach my $line (@messages) {
            my $tmp = $line;
            chomp $tmp;
            print "$tmp\n";
        }
        die "\n";
    }
}

#####
# Main
#####
if (getopts('hS:b:v', \%opts)) {
    usage() if (defined($opts{h}));

    if (defined($opts{S}) && $opts{S} =~ /\^d+$/) {
        $ssid = $opts{S};
    } else {
        usage("Saveset ID must be positive integer.\n");
    }
}
```

```
        if (defined($opts{v})) {
            $verbose = 1;
        }

        if (defined($opts{b})) {
            $pool = $opts{b};
        }
    }

    # Assemble mminfo query.
    my $query = "mminfo -q \"ssid=$ssid";
    if ($pool ne "(undef)") {
        $query .= ",pool=$pool\"";
    } else {
        $query .= "\"";
    }

    $query .= " -r volume,first,last -xc,";
    ($verbose) && print "Query: $query\n\n";

    if (open(MMI,"$query 2>&1 |")) {
        <MMI>;
        while (<MMI>) {
            my $tmp = $_;
            chomp $tmp;
            my @entries = split(/,/, $tmp);
            my $volume = $entries[0];
            my $first = Math::BigInt->new($entries[1]);
            my $last = Math::BigInt->new($entries[2]);

            $fragments{$first}{volume} = $volume;
            $fragments{$first}{end} = $last;
        }
        close(MMI);
    } else {
        die "Could not execute: $query\n";
    }

    my @volumes = ();
    my $volCount = 0;
    foreach my $fragment (sort {$a <=> $b} keys %fragments) {
        ($verbose) && print "$fragment:
$fragments{$fragment}{volume} ->
$fragments{$fragment}{end}\n";
        push(@volumes,$fragments{$fragment}{volume});
        $volCount++;
    }

    my $volString = "$volCount volume";
    $volString .= "s" if ($volCount > 1);

    print "\n$volString to be scanned in the following order:\n"
    . join(", ", @volumes) . "\n";
```

You can also download the following script from the following URL:

<http://nsrd.info/utils/volume-order.zip>

The logic for this script is as follows:

- Get from the user the saveset ID to provide scanning order for (and optionally the pool it is written in – for use when there are multiple copies)⁹;
- Run an mminfo query to determine the start/finish offset bytes for saveset fragments across all volumes;
- Output a list of volumes, ordered by the start and finish offset bytes.

Optionally, if verbose mode is enabled, the query run is shown, as is the construction of the ordering data.

Note that this script requires the use of the Perl *Math* module to avoid any risk of overflow on numbers, given all values output by mminfo are in bytes.

Output from this script can resemble the following:

```
[root@tara ~]# ./volume-order.pl -S 2680958137

57 volumes to be scanned in the following order:
800843L4, 800844L4, 800848L4, 800846L4, 800847L4, 800849L4, 800851L4, 800850L4,
800853L4, 800854L4, 800852L4, 800856L4, 800855L4, 800857L4, 800860L4, 800859L4,
800858L4, 800863L4, 800862L4, 800861L4, 800865L4, 800864L4, 800866L4, 800868L4,
800867L4, 800869L4, 800871L4, 800872L4, 800870L4, 800875L4, 800873L4, 800874L4,
800878L4, 800877L4, 800876L4, 800881L4, 800879L4, 800880L4, 800884L4, 800882L4,
800883L4, 800887L4, 800886L4, 800885L4, 800890L4, 800889L4, 800888L4, 800893L4,
800892L4, 800891L4, 800894L4, 800896L4, 800895L4, 800899L4, 800898L4, 800897L4,
800902L4
[root@tara ~]#
```

Figure 26: Sample volume-order output

When run in verbose mode, the output starts with the assembly of the ordering information:

```
[root@tara ~]# ./volume-order.pl -S 2680958137 -v
Query: mminfo -q "ssid=2680958137" -r volume,first,last -xc,

0: 800843L4 -> 2101914319
2101914320: 800844L4 -> 4861776151
4861776152: 800848L4 -> 7622421367
7622421368: 800846L4 -> 10382544327
10382544328: 800847L4 -> 13142667287
13142667288: 800849L4 -> 15903312503
15903312504: 800851L4 -> 18663174335
18663174336: 800850L4 -> 21423558423
21423558424: 800853L4 -> 24183942511
24183942512: 800854L4 -> 26943804343
26943804344: 800852L4 -> 29704449559
29704449560: 800856L4 -> 32464833647
32464833648: 800855L4 -> 35224695479
35224695480: 800857L4 -> 37985601823
```

Figure 27: Sample volume-order verbose output

⁹ Differentiating between multiple copies within the *same* pool is left as an exercise for the reader.

8.7.2 Example: mminfo2html

The XML output from mminfo is quite useful, but doesn't lend itself for immediate interpretation and display. While a browser will open the file, for instance, without additional style information, it won't necessarily display it in a tabular format.

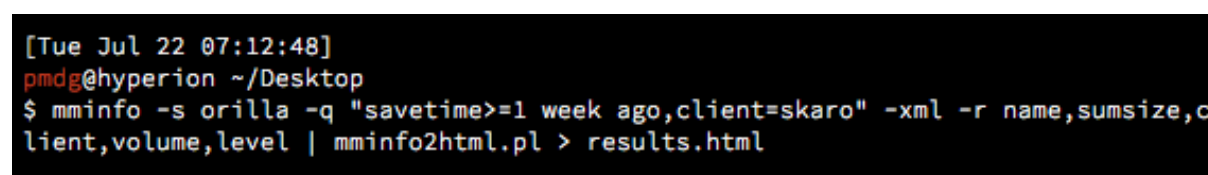
One way around this is to push the XML output into a simple Perl script that re-interprets it to HTML. In the simplest form, such a script might take the output from mminfo as standard input, and write the HTML to standard output. Thus, invoking such a command might resemble the following:

```
# mminfo -q querySpec -r reportSpec -xml | mminfo2html.pl >
file.html
```

Where:

- *querySpec* is the mminfo query specification
- *reportSpec* is the mminfo report specification
- *file.html* is the destination file to write to

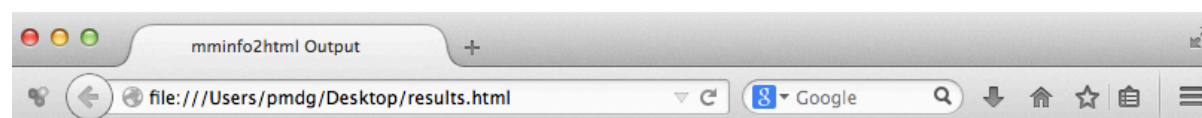
For instance:



```
[Tue Jul 22 07:12:48]
pmdg@hyperion ~/Desktop
$ mminfo -s orilla -q "savetime>=1 week ago,client=skaro" -xml -r name,sumsize,c
lient,volume,level | mminfo2html.pl > results.html
```

Figure 28: Sample execution of custom Perl script, 'mminfo2html'

The HTML file rendered might look like the following in the browser:



mminfo2html Output

Name	Sum-size	Client	Volume	Level
C:\	1168 MB	skaro	Standard.01	incr
C:\	113 MB	skaro	Standard.01	incr
DISASTER_RECOVERY:\	20 KB	skaro	Standard.01	incr
DISASTER_RECOVERY:\	20 KB	skaro	Standard.01	incr
WINDOWS ROLES AND FEATURES:\	9303 KB	skaro	Standard.01	full
WINDOWS ROLES AND FEATURES:\	9303 KB	skaro	Standard.01	full
\\?GLOBALROOT\Device\Harddisk Volume2\	30 MB	skaro	Standard.01	full
\\?GLOBALROOT\Device\Harddisk Volume2\	30 MB	skaro	Standard.01	full

Figure 29: Sample output from mminfo2html

The actual Perl script used to render this (named as mminfo2html.pl) is as follows:

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

use strict;

my @order = ();
my %entries = ();
```

```
my $count = 0;
my $reportStarted = 0;
my $haveOrder = 0;
while (<>) {
    my $line = $_;
    chomp $line;

    if ($line =~ /mminfo-tabular-report/) {
        $reportStarted = 1;
        next;
    }

    next if (!$reportStarted);

    if ($line =~ /^<result>/) {
        # Start of a result entry.
        my $nextLine = "";
        while ($nextLine ne "</result>") {
            my $nextLine = <>;
            chomp $nextLine;
            if ($nextLine =~ "</result>") {
                $haveOrder = 1;
                $count++;
                last;
            }
            if ($nextLine =~ /^<(.*?)>(.*?)<\/.*>$/) {
                my $field = $1;
                my $data = $2;
                $entries{$count}{$field} = $data;
                if (!$haveOrder) {
                    push(@order,$field);
                }
            }
        }
    }
}

print <<EOF;
<HTML>
    <HEAD>
    <TITLE>mminfo2html Output</TITLE>
    </HEAD>

    <BODY>
    <H1>mminfo2html Output</H1>
    <TABLE WIDTH=100% BORDER=1>
        <TR>
EOF

    foreach my $field (@order) {
        print "\t\t<TH><B>" . ucfirst($field) . "</B></TH>\n"
    }
    print "\t</TR>\n";

    foreach my $entry (sort {$a cmp $b} keys %entries) {
        print "\t<TR>\n";
        foreach my $field (@order) {
            print "\t<TD>" . $entries{$entry}{$field} .
"</TD>\n";
        }
        print "\t</TR>\n";
    }
print <<EOF;
```



```
</TABLE>
</BODY>
</HTML>
EOF
```

A more comprehensive version of the `mminfo2html.pl` script is available from the website at <http://nsrd.info/utils/mminfo2html.zip>. The version on the website includes such features as options to name the report, a default form of CSS for better display, and options to include a custom CSS file.

9 nsrinfo

The ‘`nsrinfo`’ command is not as frequently used as `mminfo`, yet still performs a powerful function: it allows you to query and view the content of browseable file indices.

The simplest form of `nsrinfo` is to provide a client name and the *nsavetime* of a specific backup in order to list the files that were backed up. Consider, for instance, the following sequence of commands:

```
[root@orilla ~]# mminfo -q "client=mondas,savetime>=12 hours ago"
volume      client      date        size        level  name
Slow.01     mondas      22/07/14    4501 KB     incr   /
Slow.01     mondas      22/07/14     4 B        incr   /boot
Slow.01     mondas      22/07/14     4 B        incr   /d/01
Slow.01     mondas      22/07/14     4 B        incr   /d/backup1
[root@orilla ~]# mminfo -q "client=mondas,savetime>=12 hours ago,name=/" -r nsavetime
1406026959
[root@orilla ~]# nsrinfo -t 1406026959 mondas
scanning client 'mondas' for savetime 1406026959(Tue 22 Jul 2014 21:02:39 EST) from the backup namespace
/var/log/audit/audit.log
/var/log/audit/
/var/log/cron
/var/log/maillog
/var/log/
/var/run/
/var/spool/clientmqueue/qfs6I1cXxm025206
/var/spool/clientmqueue/qfs6LNcYxi011279
/var/spool/clientmqueue/qfs6JIcXxi004020
/var/spool/clientmqueue/qfs6J9cXxc029906
/var/spool/clientmqueue/qfs6I0cXxZ025087
/var/spool/clientmqueue/qfs6I7cX0F026387
```

Figure 30: `nsrinfo`, a first look

This sequence worked as follows:

- The first `mminfo` command allowed us determine the saveset we were interested in;
- The second `mminfo` command extracted the *nsavetime* value for that particular saveset;
- The `nsrinfo` command then extracted the list of files in the client index for the nominated client against the specific *nsavetime* extracted from the previous `mminfo` command.

9.1 Command Line Arguments

The *nsrinfo* command accepts the following arguments:

- **-v** – Verbose: Includes file type, savetime, etc.
- **-V** – Alternate verbose: Includes offset in the saveset to the file, size within the saveset, namespace of application that generated the saveset, and so on.
- **-s server** – Run nsrinfo against the nominated server.
- **-L** – Run nsrinfo against the index files (useful if the server is shutdown).
- **-n** – Query against a specific namespace. Valid namespaces are:
 - *all* – All backup types
 - *bbb* – Block based backup data
 - *migrated* – NetWorker migrated savesets
 - *archive* – NetWorker archive savesets
 - *db2* – IBM DB/2 backups
 - *informix* – Informix database backups
 - *msexch* – Microsoft Exchange Data
 - *mssql* – Microsoft SQL Server Data
 - *oracle* – Oracle database backups
 - *notes* – Lotus Notes backups
 - *saphana* – SAP HANA backups
 - *sybase* – Sybase backups
- **-N filename** – Specify an exact filename to search the index for.
- **-t time** – Search for backups performed at an exact time.
- **-T** – Give filenames backed up but excludes ‘continuation’ directories.
- **-X application** – Restricts data to a specific application type, which can be one of *all*, *Informix* or *none*.
- **-x exportSpec** – Uses a specific export specification, in the same format for *mminfo*; i.e., *-xml* for XML output, *-xc*, for comma separated output, etc.

9.2 Listing files by time

The nsrinfo command we used earlier was:

```
# nsrinfo -t 1406026959 mondas
```

The reason the *nsavetime* value is often used with nsrinfo is the simple reason that it is the most exact way of specifying the actual time of the backup executed that is to be queried. If *nsavetime* isn’t used, the time specified to nsrinfo must be accurate *to the second* in order to have nsrinfo correctly target a saveset.

For instance, at the start of the output to the above command, *nsrinfo* states:

```
[root@orilla ~]# nsrinfo -t 1406026959 mondas | more
scanning client 'mondas' for savetime 1406026959(Tue 22 Jul 2014 21:02:39 EST) f
rom the backup namespace
/var/log/audit/audit.log
/var/log/audit/
/var/log/cron
/var/log/maillog
/var/log/
/var/run/
/var/spool/clientmqueue/qfs6I1cXxm025206
/var/spool/clientmqueue/qfs6LNcYxi011279
/var/spool/clientmqueue/qfs6JIcXxi004020
/var/spool/clientmqueue/qfs6J9cXxc029906
/var/spool/clientmqueue/qfs6I0cXxZ025087
```

Figure 31: Isolating saveset times in nsrinfo (1)

As the `-t` option for `nsrinfo` accepts any time format that is recognised by NetWorker, we could run the command, replacing the *nsavetime* with the human readable time expressed in brackets in the output of the command:

```
[root@orilla ~]# nsrinfo -t "Tue 22 Jul 2014 21:02:39 EST" mondas | more
scanning client 'mondas' for savetime 1406026959(Tue 22 Jul 2014 21:02:39 EST) f
rom the backup namespace
/var/log/audit/audit.log
/var/log/audit/
/var/log/cron
/var/log/maillog
/var/log/
/var/run/
/var/spool/clientmqueue/qfs6I1cXxm025206
/var/spool/clientmqueue/qfs6LNcYxi011279
/var/spool/clientmqueue/qfs6JIcXxi004020
/var/spool/clientmqueue/qfs6J9cXxc029906
```

Figure 32: Isolating saveset times in `nsrinfo` (2)

However, re-running the command and dropping the seconds from the time, or even altering them by one second in either direction yields a failure every time:

```
[root@orilla ~]# nsrinfo -t "Tue 22 Jul 2014 21:02 EST" mondas
scanning client 'mondas' for savetime 1406026920(Tue 22 Jul 2014 21:02:00 EST) f
rom the backup namespace
0 objects found
[root@orilla ~]# nsrinfo -t "Tue 22 Jul 2014 21:02:40 EST" mondas
scanning client 'mondas' for savetime 1406026960(Tue 22 Jul 2014 21:02:40 EST) f
rom the backup namespace
0 objects found
[root@orilla ~]# nsrinfo -t "Tue 22 Jul 2014 21:02:38 EST" mondas
scanning client 'mondas' for savetime 1406026958(Tue 22 Jul 2014 21:02:38 EST) f
rom the backup namespace
0 objects found
```

Figure 33: Isolating saveset times in `nsrinfo` (3)

For this reason, since the *exact* time must be extracted, it is usually more convenient to provide this time in *nsavetime* format to `nsrinfo`.

9.3 Finding previously backed up files

Another aspect of `nsrinfo` is its capability, if provided with the exact path to a filename, of quickly providing a list of backup versions of that file. This same functionality is more familiar to users within the recovery interface, such as:

```
[root@orilla ~]# recover
Current working directory is /root/
recover> versions /usr/sbin/nsradmin

Versions of '/usr/sbin/nsradmin':

3428 -rwxr-xr-x root      root      3509208 May 29 21:05 nsradmin*
      save time:  Sat 12 Jul 2014 21:00:02 EST
      location:   Slow.01 at Slow-01

3276 -rwxr-xr-x root      root      3350648 Apr 17 07:35 nsradmin*
      save time:  Sun 29 Jun 2014 21:00:02 EST
      location:   Slow.01 at Slow-01
```

Figure 34: Identifying backup versions via *recover*

Via *nsrinfo*, the process is as follows:

```
# nsrinfo -N /path/to/file clientName
```

```
[root@orilla ~]# nsrinfo -N /usr/sbin/nsradmin orilla
scanning client 'orilla' for all savetimes from the backup namespace
/usr/sbin/nsradmin, date=1405162802 Sat 12 Jul 2014 21:00:02 EST
/usr/sbin/nsradmin, date=1404039602 Sun 29 Jun 2014 21:00:02 EST
```

Figure 35: Identifying backup versions via *nsrinfo*

The key difference between the output of *nsrinfo* and the output of *recover* in this scenario is that *recover* also sifts information from the media database to elaborate on volume details for the file.

Using the verbose options for *nsrinfo* in this type of query can provide additional information about a file backed up, in much the same way as *recover* does:

```
# nsrinfo -N /path/to/file clientName -vV
```

For instance:

```
[root@orilla ~]# nsrinfo -N /usr/sbin/nsradmin orilla -vV
scanning client 'orilla' for all savetimes from the backup namespace
UNIX ASDF v2 file '/usr/sbin/nsradmin', size=1355768, off=649745680, app=backup(
1), date=1405162802 Sat 12 Jul 2014 21:00:02 EST, fid = 64770.3419088, file size
=3509208, mtime=1401361542 Thu 29 May 2014 21:05:42 EST, ctime=1405145263 Sat 12
Jul 2014 16:07:43 EST, access permissions=-rwxr-xr-x, uid=0(root), gid=0(root)
UNIX ASDF v2 file '/usr/sbin/nsradmin', size=1296016, off=1935470940, app=backup
(1), date=1404039602 Sun 29 Jun 2014 21:00:02 EST, fid = 64770.3419079, file siz
e=3350648, mtime=1397684159 Thu 17 Apr 2014 07:35:59 EST, ctime=1402525379 Thu 1
2 Jun 2014 08:22:59 EST, access permissions=-rwxr-xr-x, uid=0(root), gid=0(root)
```

Figure 36: Using *nsrinfo* with the verbose flags

You can, if you wish, do simple backup file finding using *nsrinfo*. One such mechanism is to make use of the option to scan *all* indices for a client by not specifying a time at all:

```
# nsrinfo clientName
```

The output in this scenario will resemble the following:

```
[root@orilla ~]# nsrinfo mondas | more
scanning client 'mondas' for all savetimes from the backup namespace
/var/log/audit/audit.log, date=1406026959 Tue 22 Jul 2014 21:02:39 EST
/var/log/audit/, date=1406026959 Tue 22 Jul 2014 21:02:39 EST
/var/log/cron, date=1406026959 Tue 22 Jul 2014 21:02:39 EST
/var/log/maillog, date=1406026959 Tue 22 Jul 2014 21:02:39 EST
/var/log/, date=1406026959 Tue 22 Jul 2014 21:02:39 EST
/var/run/, date=1406026959 Tue 22 Jul 2014 21:02:39 EST
/var/spool/clientmqueue/qfs6I1cXxm025206, date=1406026959 Tue 22 Jul 2014 21:02:
39 EST
/var/spool/clientmqueue/qfs6LNcYxi011279, date=1406026959 Tue 22 Jul 2014 21:02:
39 EST
/var/spool/clientmqueue/qfs6JIcXxi004020, date=1406026959 Tue 22 Jul 2014 21:02:
39 EST
```

Figure 37: Using nsrinfo to view all files in a client index

As can be imagined, the verbosity of this command is highly variable, and will be extreme in situations where a client has a large number of files in its backup indices. Yet, completely searching client indices for a particular file is no more complex than running the above nsrinfo command and then searching the results. On Linux/Unix servers, that can be as simple as:

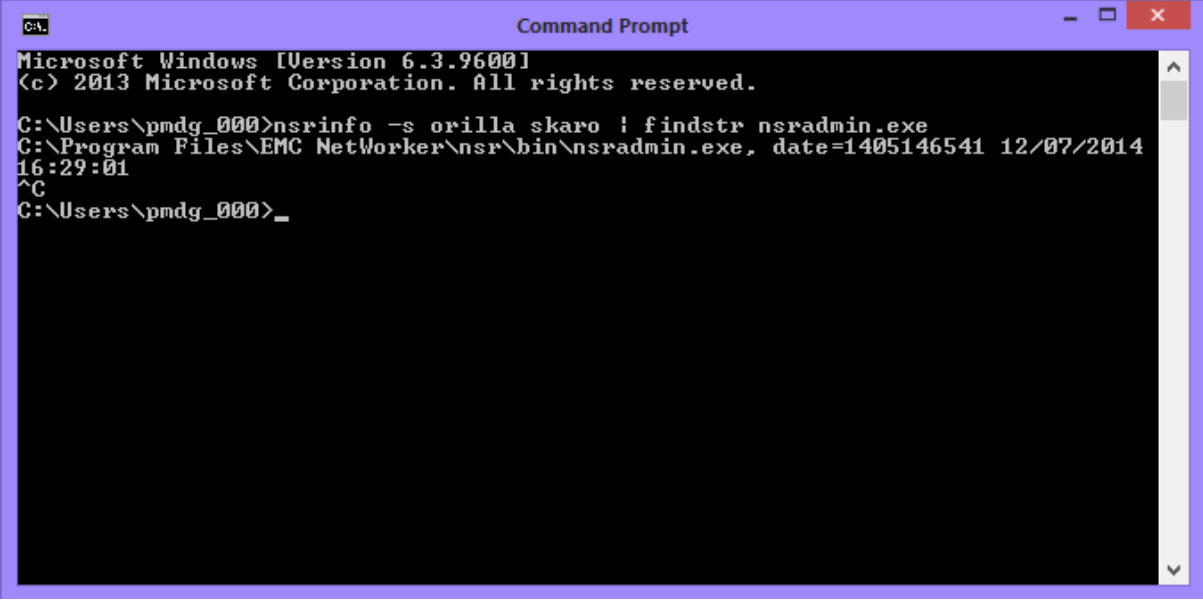
```
# nsrinfo clientName | grep file
```

```
[root@orilla ~]# nsrinfo mondas | grep vpn-iptables.sh
/root/iptables/vpn-iptables.sh, date=1404041715 Sun 29 Jun 2014 21:35:15 EST
```

Figure 38: Using nsrinfo on Unix to search for backed up files

On Windows, the alternative would be to use *findstr* as part of the command sequence:

```
C:\> nsrinfo clientName | findstr file
```



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\pmdg_000>nsrinfo -s orilla skaro | findstr nsradmin.exe
C:\Program Files\EMC NetWorker\nsr\bin\nsradmin.exe, date=1405146541 12/07/2014
16:29:01
^C
C:\Users\pmdg_000>_
```

Figure 39: Using nsrinfo on Windows to search for backed up files

Keep in mind with this technique that extracting *all* the index details for a client, even when immediately piping it to a search routine, could impact backup server performance. For example, consider a moderately sized corporate fileserver with a million files – if there's also a standard backup process of say, weekly fulls and daily incrementals, with short-term backups retained for 6 weeks and long-term backups retained for 10 years, the number of files that would be output from

this version of mminfo would be mammoth. Assuming a 5% change on incrementals this would equate to:

- 6 x 1,000,000 for weekly full backups
- 10 x 12 x 1,000,000 for monthly full backups (assuming a worst case scenario)
- 6 x 6 x (1,000,000 * 0.05) for daily incremental backups
- Totalling 127,800,000 files.

Left as an exercise for the reader, a more intelligent and less resource intensive solution would be to use the following program logic:

- Accept as input:
 - A client name
 - A start date
 - An end date
 - A file path/name
- Query from mminfo all *nsavetime* times for:
 - The client nominated
 - Between the start and end date
- Loop through each *nsavetime* and run nsrinfo against it:
 - Searching for the file path/name specified
 - Output any matching entries

While the NetWorker Management Console now includes a file search option, the advantage of a scripted file search is that it can be executed at any time. Taking a DevOps approach, this would allow for situations where:

1. The script is made available via a web portal;
2. The script is made available for auditors;
3. The script is made available for eDiscovery purposes.

Obviously in each scenario it would be important to secure access to the script – for a multi-tenanted backup solution, for instance (such as in (1), above), additional logic might be introduced to prevent someone from entering an arbitrary client name, and instead making the client name a selection based on appropriate criteria.

10 **gstclreport**

Technically, *gstclreport* is not a core NetWorker reporting tool. Instead, it stems from NetWorker Management Console (NMC), which in theory can serve multiple NetWorker servers.

gstclreport is located in the NMC bin directory; on Unix, that will typically be `/opt/lgtonmc/bin`. On Windows systems, the default install path is `C:\Program Files\EMC NetWorker\Management\bin`.

Running *gstclreport* requires an appropriately configured and accessible Java environment. Running it without suitable Java access will result in the following:


```
[root@orilla ~]# /opt/lgtonmc/bin/gstclreport
% JAVA_HOME is not set.
% Java Runtime Environment 1.6 is required to run this script.
% Please read comments at the top of gstclreport file to set
% JAVA_HOME to a valid JRE location.
[root@orilla ~]#
```

Figure 40: Running `gstclreport` without an accessible Java environment

Before running `gstclreport` it is necessary to set a valid `JAVA_HOME` environment, and it is also likely that the path to `gstclreport` is not in the default system executable path. For example, on CentOS Linux, one might prepare a shell for `gstclreport` commands by setting up:

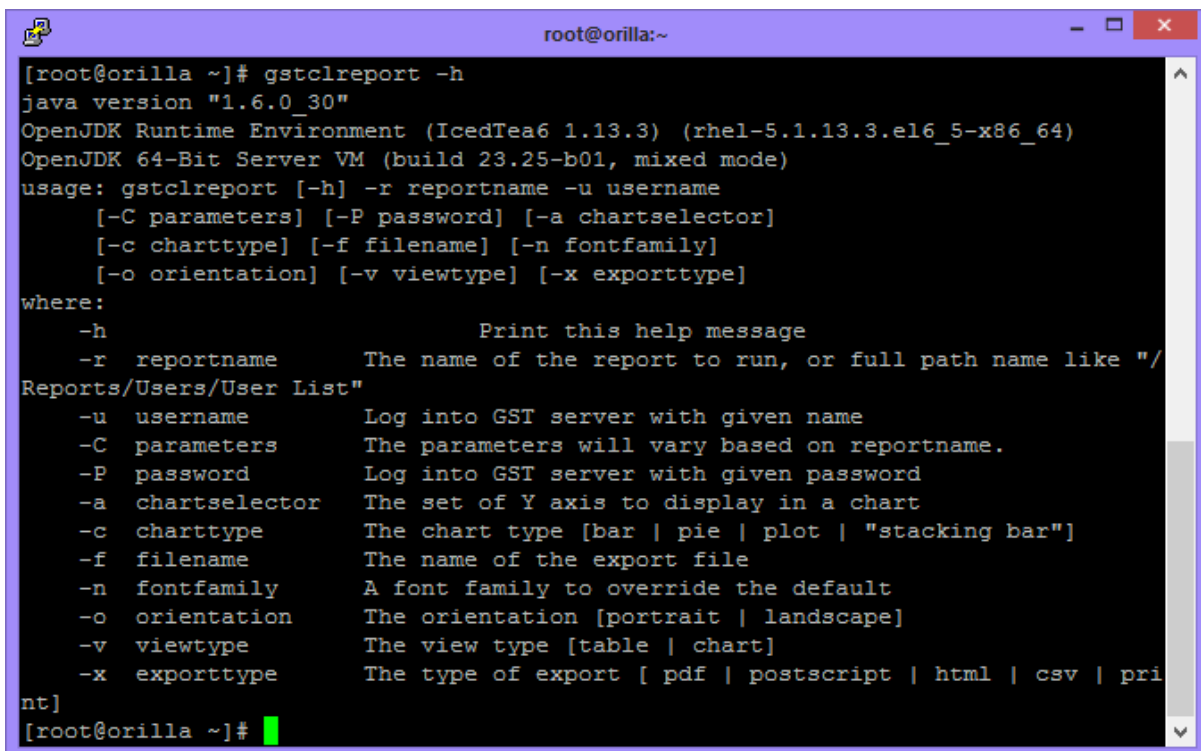
```
export PATH=$PATH:/opt/lgtonmc/bin
export JAVA_HOME=/usr/lib/jvm/java
```

On a Windows system, the command shell sequence might resemble:

```
set JAVA_HOME=C:\Program Files\java\jre7
set PATH=%PATH%;C:\Program Files\EMC
NetWorker\Management\GSTD\bin
```

(The line starting “set PATH” continues through to and includes the text “...GSTD\bin”.)

With the correct PATH and `JAVA_HOME` settings established, the output from `gstclreport` will resemble the following:



```
root@orilla:~
[root@orilla ~]# gstclreport -h
java version "1.6.0_30"
OpenJDK Runtime Environment (IcedTea6 1.13.3) (rhel-5.1.13.3.el6_5-x86_64)
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)
usage: gstclreport [-h] -r reportname -u username
        [-C parameters] [-P password] [-a chartselector]
        [-c charttype] [-f filename] [-n fontfamily]
        [-o orientation] [-v viewtype] [-x exporttype]
where:
  -h                Print this help message
  -r reportname     The name of the report to run, or full path name like
"/Reports/Users/User List"
  -u username       Log into GST server with given name
  -C parameters     The parameters will vary based on reportname.
  -P password       Log into GST server with given password
  -a chartselector  The set of Y axis to display in a chart
  -c charttype      The chart type [bar | pie | plot | "stacking bar"]
  -f filename       The name of the export file
  -n fontfamily     A font family to override the default
  -o orientation    The orientation [portrait | landscape]
  -v viewtype       The view type [table | chart]
  -x exporttype     The type of export [ pdf | postscript | html | csv | pri
nt]
[root@orilla ~]#
```

Figure 41: Running `gstclreport` with `JAVA_HOME` correctly established

There are several distinct advantages of NMC reports:

- Their data is retained independently of the NetWorker indices and media databases, and therefore can report on events not currently in the NetWorker databases;
- They can provide summary information that spans multiple NetWorker servers;

- They are designed with a considerably more graphical approach, and are more appropriate for inclusion into regular management or auditor reporting.

In particular, one of the more common scenarios backup administrators are requested to report on is *failed* backups. These typically *aren't* included in the NetWorker media database, as it only retains information on backups that have completed¹⁰.

We'll start with the simplest invocations of `gstclreport`, using the following arguments:

- `-u username` – Specify the accessing NMC user account for running the report
- `-r report` – Specify the report name to run. This should optimally be specified in the full path to the report
- `-f filename` – Specify the output filename
- `-x format` – Specify the export format (pdf, postscript, html, csv, print)

Using the report path example specified in the default output from `gstclreport`, the command might resemble the following:

```
# gstclreport -u administrator -r "/Reports/Users/User List"
-f userlist.pdf -x pdf
```

When run, `gstclreport` will prompt for the password for the specified user (in this case, 'administrator'), and the execution sequence will resemble the following:

```
[root@orilla ~]# gstclreport -u administrator -r "/Reports/Users/User List" -f userlist.pdf -x pdf
java version "1.6.0_32"
OpenJDK Runtime Environment (IcedTea6 1.13.4) (rhel-6.1.13.4.el6_5-x86_64)
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)
Enter Password:
Generated report "User List" as file userlist.pdf
[root@orilla ~]#
```

Figure 42: Generating a basic report out of `gstclreports`

An additional argument available to `gstclreport` is `-P password`, which allows the specification of the NMC login password for the nominated user. This option should be used with caution, and should never be used with the administrator account for NMC. When this option is used, the password is visible to anyone who is able to perform a full process listing on the host the command is run on – for Unix and Linux systems, this will typically be revealed to anyone running “`ps -eaf`” at the time the command is executing. (Comparable techniques are available for Windows.)

If the password option is to be used for fully automating the generation of reports, one of the two following techniques should be used:

- Create a dedicated ‘reports’ account in NMC without administrative privileges
- Use a scripting language such as *Expect*, which allows the scripting of automated interactive processes

Many of the reports available in NMC have a variety of options available; these are accessible via the `-C` option. The options available will vary by each report, and they can be viewed by running the `gstclreport` with each of the options `-h` (help), `-C`, `-u username` and `-r reportName`. This will result in `gstclreport` printing all available options for the selected report. For instance:

```
# gstclreport -u username -h -r "/Reports/reportpath" -C
```

¹⁰ A notable exception being that failed tape-based backups may be retained in the database until the tape is recycled. This is declining in usefulness as disk based backups increase in adoption.

For instance:

```
[root@orilla ~]# gstclreport -u administrator -h -r "/Reports/Users/User Audit"
-C
java version "1.6.0_32"
OpenJDK Runtime Environment (IcedTea6 1.13.4) (rhel-6.1.13.4.el6_5-x86_64)
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)
Enter Password:
Missing argument for option - C
usage: gstclreport [-h] -r reportname -u username
        [-P password] [-a chartselector] [-c charttype]
        [-f filename] [-n fontfamily] [-o orientation]
        [-v viewtype] [-x exporttype] [-C Category argument]
        [-C "Event Time" argument] [-C "Object Name" argument]
        [-C "Object Type" argument] [-C Operation argument]
        [-C "Server Name" argument] [-C "Server Type" argument]
        [-C "User Name" argument]
where:
  -h                Print this help message
  -r reportname      The name of the report to run, or full path name like "/
Reports/Users/User List"
  -u username        Log into GST server with given name
  -P password        Log into GST server with given password
  -a chartselector    The set of Y axis to display in a chart
  -c charttype        The chart type [bar | pie | plot | "stacking bar"]
  -f filename        The name of the export file
  -n fontfamily       A font family to override the default
  -o orientation      The orientation [portrait | landscape]
  -v viewtype         The view type [table | chart]
  -x exporttype       The type of export [ pdf | postscript | html | csv | pri
nt]
  -C Category argument  Where argument is a comma separated list of Cate
gorys
  -C "Event Time" argument  Where argument is a From and To date
  -C "Object Name" argument  Where argument is a single Object Name
  -C "Object Type" argument  Where argument is a single Object Type
  -C Operation argument  Where argument is a comma separated list of Oper
ations
  -C "Server Name" argument  Where argument is a comma separated list of Serv
er Names
  -C "Server Type" argument  Where argument is a single Server Type
  -C "User Name" argument  Where argument is a comma separated list of User
Names
```

Figure 43: Determining gstclreport configuration options

For example, to run the user audit report for *just* the server 'orilla', the command might look like the following:

```
# gstclreport -u administrator -r "/Reports/Users/User Audit"
-x pdf -f audit.pdf -C "Server Name" orilla
```

```
[root@orilla ~]# gstclreport -u administrator -r "/Reports/Users/User Audit" -x
pdf -f audit.pdf -C "Server Name" orilla
java version "1.6.0_30"
OpenJDK Runtime Environment (IcedTea6 1.13.3) (rhel-5.1.13.3.el6_5-x86_64)
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)
Enter Password:
Generated report "User Audit" as file audit.pdf
[root@orilla ~]#
```

Figure 44: Generating a report with gstclreport and custom configuration options

Many of the reports in NMC (and therefore via gstclreport) can take a date or date range output. Consider, for instance, the “Daily Summary” report. To determine what arguments it takes, run the report with both the -C and the -h options:

```
# gstclreport -u administrator -r "/Reports/NetWorker Backup
Status/Daily Summary" -x pdf -f last_7_days.pdf -h -C
```

This will output the full options available:

```
[root@orilla ~]# gstclreport -u administrator -r "/Reports/NetWorker Backup Stat
us/Daily Summary" -x pdf -f last_7_days.pdf -h -C
java version "1.6.0_30"
OpenJDK Runtime Environment (IcedTea6 1.13.3) (rhel-5.1.13.3.el6_5-x86_64)
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)
Enter Password:
Missing argument for option - C
usage: gstclreport [-h] -r reportname -u username
        [-P password] [-a chartselector] [-c charttype]
        [-f filename] [-n fontfamily] [-o orientation]
        [-v viewtype] [-x exporttype] [-C "Group Name" argument]
        [-C "Group Start Time" argument] [-C "Server Name" argument]
        [-C Status argument]
where:
  -h                                Print this help message
  -r reportname                    The name of the report to run, or full path name like "/"
Reports/Users/User List"
  -u username                      Log into GST server with given name
  -P password                      Log into GST server with given password
  -a chartselector                The set of Y axis to display in a chart
  -c charttype                    The chart type [bar | pie | plot | "stacking bar"]
  -f filename                     The name of the export file
  -n fontfamily                   A font family to override the default
  -o orientation                  The orientation [portrait | landscape]
  -v viewtype                     The view type [table | chart]
  -x exporttype                   The type of export [ pdf | postscript | html | csv | pri
nt]
  -C "Group Name" argument        Where argument is a comma separated list of Grou
p Names
  -C "Group Start Time" argument  Where argument is a From and To date
  -C "Server Name" argument       Where argument is a comma separated list of Serv
er Names
  -C Status argument              Where argument is a comma separated list of Stat
uss
[root@orilla ~]#
```

Figure 45: Dealing with date ranges in gstclreport (i)

To specify a group start time range, it's important to first understand what date/time format the server is going to expect. You can get prompted on this by running the command with a nonsensical date/timestamp – e.g., an invalid start time of 88:88:88:

```
# gstclreport -u administrator -r "/Reports/NetWorker Backup  
Status/Daily Summary" -x pdf -f last_7_days.pdf -C "Group  
Start Time" "88:88:88"
```

When run, this will trigger the following output:

```
[root@orilla ~]# gstclreport -u administrator -r "/Reports/NetWorker Backup Stat  
us/Daily Summary" -x pdf -f last_7_days.pdf -C "Group Start Time" "88:88:88"  
java version "1.6.0_32"  
OpenJDK Runtime Environment (IcedTea6 1.13.4) (rhel-6.1.13.4.el6_5-x86_64)  
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)  
Enter Password:  
Could not parse the date time value "88:88:88"  
Please use enter date in this format. "d/MM/yy h:mm:ss a" i.e. 25/07/14 6:58:36  
AM  
usage: gstclreport [-h] -r reportname -u username  
        [-P password] [-a chartselector] [-c charttype]  
        [-f filename] [-n fontfamily] [-o orientation]  
        [-v viewtype] [-x exporttype] [-C "Group Name" argument]  
        [-C "Group Start Time" argument] [-C "Server Name" argument]  
        [-C Status argument]  
where:  
  -h                      Print this help message  
  -r reportname           The name of the report to run, or full path name like "  
Reports/Users/User List"  
  -u username             Log into GST server with given name  
  -P password             Log into GST server with given password  
  -a chartselector        The set of Y axis to display in a chart  
  -c charttype            The chart type [bar | pie | plot | "stacking bar"]  
  -f filename             The name of the export file  
  -n fontfamily           A font family to override the default  
  -o orientation          The orientation [portrait | landscape]  
  -v viewtype             The view type [table | chart]  
  -x exporttype           The type of export [ pdf | postscript | html | csv | pri  
nt]  
  -C "Group Name" argument Where argument is a comma separated list of Grou  
p Names  
  -C "Group Start Time" argument Where argument is a From and To date  
  -C "Server Name" argument Where argument is a comma separated list of Serv  
er Names  
  -C Status argument      Where argument is a comma separated list of Stat  
uss
```

Figure 46: Dealing with date ranges in gstclreport (2)

Directly before the help output, gstclreport has stated:

```
Could not parse the date time value "88:88:88"  
Please use enter [sic] date in this format. "d/MM/yy h:mm:ss  
a" i.e. "25/07/14 6:58:36 AM"
```

Generally, if gstclreport states that a time is to be included in the format, it's optional. In this case, the report can be run as follows:

```
# gstclreport -u administrator -r "/Reports/NetWorker Backup  
Status/Daily Summary" -x pdf -f last_7_days.pdf -C "Group  
Start Time" "19/07/14" "25/07/14"
```

The output from this will resemble the following:

```
[root@orilla ~]# gstclreport -u administrator -r "/Reports/NetWorker Backup Stat  
us/Daily Summary" -x pdf -f last_7_days.pdf -C "Group Start Time" "19/07/14" "25  
/07/14"  
java version "1.6.0_32"  
OpenJDK Runtime Environment (IcedTea6 1.13.4) (rhel-6.1.13.4.el6_5-x86_64)  
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)  
Enter Password:  
Generated report "Daily Summary" as file last_7_days.pdf  
[root@orilla ~]#
```

Figure 47: Dealing with date ranges in `gstclreport` (3)

You'll note in the command that a full date *range* was specified – in this instance, we're looking for any group that started between 19 July 2014 and 25 July 2014. If instead you wanted to generate the report for all backups *since* a particular date, you can just include the start date. For example:

```
# gstclreport -u administrator -r "/Reports/NetWorker Backup  
Status/Daily Summary" -x pdf -f july14_backups.pdf -C "Group  
Start Time" "1/7/14"
```

In this scenario, the report will pick up all backups run from 1 July 2014:

```
[root@orilla ~]# gstclreport -u administrator -r "/Reports/NetWorker Backup Stat  
us/Daily Summary" -x pdf -f july14_backups.pdf -C "Group Start Time" "1/7/14"  
java version "1.6.0_32"  
OpenJDK Runtime Environment (IcedTea6 1.13.4) (rhel-6.1.13.4.el6_5-x86_64)  
OpenJDK 64-Bit Server VM (build 23.25-b01, mixed mode)  
Enter Password:  
Generated report "Daily Summary" as file july14_backups.pdf  
[root@orilla ~]#
```

Figure 48: Dealing with start dates only in `gstclreport`

11 Miscellaneous reporting

While `mminfo`, `nsrinfo` and `gstclreport` represent the core reporting functions available to the NetWorker administrator, there are a few other tricks that should be in any NetWorker administrator's command line arsenal.

11.1 `nsr_render_log`

In NetWorker 7.4, EMC introduced the *raw* log format. The purpose of this was simple: to allow logs to be written in a language neutral format so they can be viewed in a *different* language from the install site if necessary.

The most common use of `nsr_render_log` resembles the following:

```
# nsr_render_log infile.raw > outfile.log
```

Where:

- *infile.raw* is the name of the log file to be read
- *outfile.log* is the name of the resulting log file.

In both instances, *infile.raw* and *outfile.log* must be specified with the full intended path to the file – otherwise NetWorker will attempt to read the file from the current directory, and your command shell will attempt to write the rendered file to the current directory as well.

While there are other ways `nsr_render_log` might be invoked, the above will be the most common. Alternately, as you may have gathered from the above syntax, another common invocation will be:

```
# nsr_render_log infile.raw | more
```

The first invocation writes the rendered log to another file; the second writes it to standard out, piping it to the *more* utility, which allows viewing on either platform.

There are a variety of other usage scenarios for `nsr_render_log` that are particularly useful for an administrator in a hurry. As you'd know, the size of the log files can be quite variable; depending on when services are restarted, some log files may grow to hundreds of megabytes or more. Sifting through all that information, particularly for one or two specific events, for a particular time period can become quite tedious. Depending on your own operating system or applications, merely *opening* a large log file, once rendered, may be problematic.

In such situations, `nsr_render_log` has a few options that can help:

- **-S *startTime*** – Only render log entries from the given start time, in any NetWorker acceptable format
- **-E *endTime*** – Stop rendering when the given end time is reached, expressed in any NetWorker acceptable format
- **-N *lineCount*** – Render *lineCount* lines and stop
- **-B *startLine*** – Start rendering at the given line number
- **-O *program*** – Render logs coming only from a specific nominated program (e.g, 'nsrmmmd')
- **-G *group*** – Render logs that have come from the nominated group
- **-J *host*** – Render logs that reference the given host (not to be confused with the server hostname option, **-H *server***)

For instance, consider the scenario where the client *miranda* has been having backup issues for the last couple of days. To view details pertaining to it from the NetWorker server `daemon.raw` file, the command used might be:

```
# nsr_render_log -J miranda -S "2 days ago" daemon.raw
```



```
[root@orilla ~]# nsr_render_log -J miranda -S "2 days ago" /nsr/logs/daemon.raw
70896 28/07/14 21:00:06 0 0 2 2752296704 2194 0 orilla.turbamentis.int nsrd NSR
info miranda:/ saving to pool 'Stage Slow' (Slow.01)
91797 28/07/14 21:00:06 4 5 0 2825950976 2517 0 orilla.turbamentis.int nsrmmd N
SR severe Unable to perform direct file save with adv_file device 'Slow-01'; set
ting up traditional save for save-set ID '4292226358' (miranda:/)
71659 28/07/14 21:31:35 0 0 2 2752296704 2194 0 orilla.turbamentis.int nsrd NSR
info miranda:/ done saving to pool 'Stage Slow' (Slow.01) 1250 MB
77562 28/07/14 21:31:35 2 5 0 443975424 3589 0 orilla.turbamentis.int savegrp N
SR warning job (352006) host: miranda savepoint: / had ERROR indication(s) at co
mpletion
90491 28/07/14 21:31:35 1 5 0 443975424 3589 0 orilla.turbamentis.int savegrp N
SR notice miranda:/ failed.
90492 28/07/14 21:31:35 1 5 0 443975424 3589 0 orilla.turbamentis.int savegrp N
SR notice miranda:/ will retry 1 more time(s).
70896 28/07/14 21:31:45 0 0 2 2752296704 2194 0 orilla.turbamentis.int nsrd NSR
info miranda:/ saving to pool 'Stage Slow' (Slow.01)
91797 28/07/14 21:31:45 4 5 0 2825950976 2517 0 orilla.turbamentis.int nsrmmd N
SR severe Unable to perform direct file save with adv_file device 'Slow-01'; set
ting up traditional save for save-set ID '4174787745' (miranda:/)
71659 28/07/14 21:57:20 0 0 2 2752296704 2194 0 orilla.turbamentis.int nsrd NSR
info miranda:/ done saving to pool 'Stage Slow' (Slow.01) 190 MB
```

Figure 49: Using `nsr_render_log` for a specific client and date/time range

The `nsr_render_log` program has a variety of other options, including:

- `-a` – Don't output the activity ID
- `-c` – Don't output the category of events.
- `-d` – Don't output timestamps
- (etc)
- `-z` – Obfuscate identifying/secure information
- `-x exportSpec` – Specify output specification (e.g., `"-xc,"` for comma separated, `"-x'c\t"` for tab separated output. Note that xml format is not applicable for `nsr_render_log`).

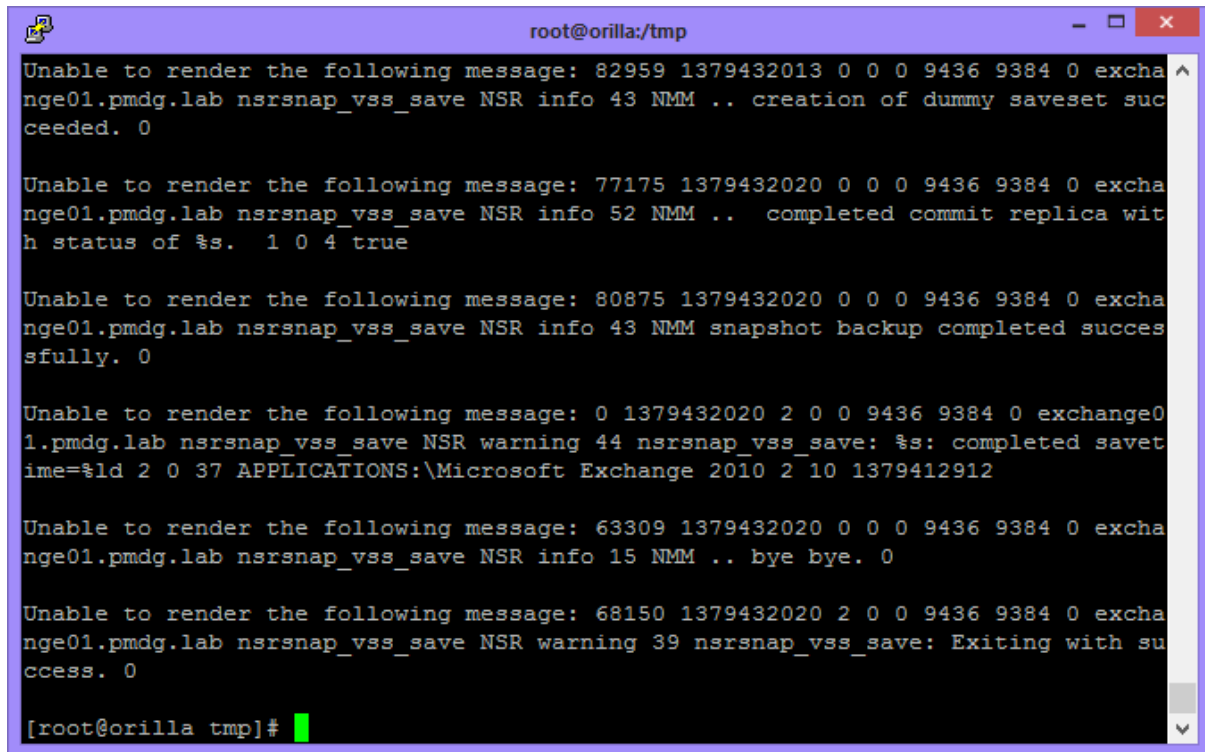
Be particularly careful of the `-z` option. While it is meant to completely replace all host names with generic hosts, it doesn't *always* do this (e.g., hostname references in index paths will still be cited).

11.1.1 Remote rendering

The `nsr_render_log` program is only capable of rendering log messages that have been produced by the NetWorker options installed on the host it is run on. This becomes relevant if you are using NetWorker modules – e.g., the NetWorker Module for Microsoft Applications or the NetWorker Module for Databases and Applications.

In these situations, if you need to render a log generated by one of these modules, you need to render it on a system with the module installed. Copying an `nmm.raw` log across to a Unix server and trying to render it there will simply result in a series of 'rendered' messages about `nsr_render_log` being unable to render the lines from the log file.

For instance, below is the final few lines of an `nmm.raw` log from an Exchange server backup, rendered on a Linux backup server:



```
root@orilla:/tmp
Unable to render the following message: 82959 1379432013 0 0 0 9436 9384 0 exchange01.pmdg.lab nsrsnap_vss_save NSR info 43 NMM .. creation of dummy saveset succeeded. 0

Unable to render the following message: 77175 1379432020 0 0 0 9436 9384 0 exchange01.pmdg.lab nsrsnap_vss_save NSR info 52 NMM .. completed commit replica with status of %s. 1 0 4 true

Unable to render the following message: 80875 1379432020 0 0 0 9436 9384 0 exchange01.pmdg.lab nsrsnap_vss_save NSR info 43 NMM snapshot backup completed successfully. 0

Unable to render the following message: 0 1379432020 2 0 0 9436 9384 0 exchange01.pmdg.lab nsrsnap_vss_save NSR warning 44 nsrsnap_vss_save: %s: completed save time=%ld 2 0 37 APPLICATIONS:\Microsoft Exchange 2010 2 10 1379412912

Unable to render the following message: 63309 1379432020 0 0 0 9436 9384 0 exchange01.pmdg.lab nsrsnap_vss_save NSR info 15 NMM .. bye bye. 0

Unable to render the following message: 68150 1379432020 2 0 0 9436 9384 0 exchange01.pmdg.lab nsrsnap_vss_save NSR warning 39 nsrsnap_vss_save: Exiting with success. 0

[root@orilla tmp]#
```

Figure 50: Rendering a log on a system without the originating module

11.1.2 Aside – Auto-rendered Log Files

NetWorker does have the option to automatically render raw files into log format on the fly, writing them concurrently as the raw files are written. (If space is a concern, this option shouldn't be used – though it could equally be argued that if space occupied by log files *is* an issue, you have bigger problems.)

Enabling auto-rendering is achieved via *nsradmin*, a tool we'll cover in greater detail later. For now, on any host you want to have logs automatically rendered on, you use the following sequence to turn on automated log rendering:

- Run *nsradmin* against the local *nsrexecd* process

```
# nsradmin -p nsrexecd -s localhost
```
- View the registered log files being managed

```
nsradmin> print type: NSR log
```
- Set a path in the runtime rendered log field for the specific log file

```
nsradmin> print type: NSR log; name: logname
nsradmin> update runtime rendered log: /path/to/logname.log
```

```
[root@orilla ~]# nsradmin -p nsrexecd -s localhost
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> print type: NSR log
      type: NSR log;
      administrator: root, "user=root,host=orilla.turbamentis.int";
      owner: NMC Log File;
      maximum size MB: 2;
      maximum versions: 10;
      runtime rendered log: ;
      runtime rollover by size: Disabled;
      runtime rollover by time: ;
      name: gstd.raw;
      log path: /opt/lgtomc/logs/gstd.raw;

      type: NSR log;
      administrator: root, "user=root,host=orilla.turbamentis.int";
      owner: NetWorker;
      maximum size MB: 2;
      maximum versions: 10;
      runtime rendered log: ;
      runtime rollover by size: Disabled;
      runtime rollover by time: ;
      name: daemon.raw;
      log path: /nsr/logs/daemon.raw;
nsradmin> print type: NSR log; name: daemon.raw
      type: NSR log;
      administrator: root, "user=root,host=orilla.turbamentis.int";
      owner: NetWorker;
      maximum size MB: 2;
      maximum versions: 10;
      runtime rendered log: ;
      runtime rollover by size: Disabled;
      runtime rollover by time: ;
      name: daemon.raw;
      log path: /nsr/logs/daemon.raw;
nsradmin> update runtime rendered log: /nsr/logs/daemon.log
      runtime rendered log: /nsr/logs/daemon.log;
Update? y
updated resource id 13.0.197.64.47.0.0.0.31.102.92.83.192.168.100.4(2)
nsradmin>
```

Figure 51: Configuring the runtime rendered log for a raw file

Once the runtime rendered log been set, it's necessary to stop and restart NetWorker services on the host to have the changes take affect. Using nsradmin will be covered in greater detail at a later point in this guide. For now, to exit nsradmin, type 'quit' at the "nsradmin>" prompt.

Note:

- If you want to automatically parse the runtime rendered logs, bear in mind there is currently a flaw with the way they are generated on Windows (to be fixed in a later NetWorker release after 8.2.x). This flaw can result in what would be multiple individual lines concatenated.

11.2 nsrsgrpcmp

Introduced in NetWorker 8 is the ability to access recent savegroup completion notifications from the command line.

Run without any arguments, nsrsgrpcmp will provide a prompt on the various options available:

```
[root@orilla ~]# nsrsgrpcmp
Usage:
    nsrsgrpcmp [ -s server ] groupname
    nsrsgrpcmp [ -s server ] -L [ groupname ]
    nsrsgrpcmp [ -s server ] [ -HNaior ] [ -b num_bytes | -l num_lines ] [
-c clientname ] [ -n jobname ] [ -t start_time ] groupname
    nsrsgrpcmp [ -s server ] -R jobid

[root@orilla ~]#
```

Figure 52: Options for nsrsgrpcmp

The most typical way of running nsrsgrpcmp is to extract the most recent savegroup completion report for a particular group. This can be done by executing:

```
# nsrsgrpcmp groupName
```

For instance:

```
[root@orilla ~]# nsrsgrpcmp Laptops
archon, savefs, "succeeded:full:savefs"
* archon:savefs savefs archon: succeeded.
archon, /, "succeeded:incr:save"
* archon:/ Warning: `/Library/Caches/CrashPlan/42/cpb00000000000000001303372/cpbmf'
changed during save
* archon:/ Expected 68610392 bytes for `/Users/pmdg/Library/Caches/com.apple.Add
ressBookSourceSync/Cache.db-wal', got 68614512 bytes
* archon:/ Warning: `/Users/pmdg/Library/Caches/com.apple.AddressBookSourceSync/
Cache.db-wal' size grew during save
* archon:/ 66135:save: NSR directive file (/nsr/cores/nsrexecd/.nsr) parsed
* archon:/ 66135:save: NSR directive file (/nsr/run/.nsr) parsed
* archon:/ 66135:save: NSR directive file (/Users/pmdg/Desktop/DNB/.nsr) parsed
* archon:/ 66135:save: NSR directive file (/Users/pmdg/Documents/.nsr) parsed
```

Figure 53: Extracting the most recent savegroup completion details with nsrsgrpcmp

To determine what savegroup completion details are available for any given group, run the command:

```
# nsrsgrpcmp -L groupName
```

```
[root@orilla ~]# nsrsgrpcmp -L "Synology Saturday"
name, start time, job id, previous job id, completion status
Synology Saturday, Sun Jul 27 22:00:00 2014(1406462400), 320528, 0, succeeded
Synology Saturday, Mon Jul 28 22:00:00 2014(1406548800), 352026, 0, succeeded
Synology Saturday, Tue Jul 29 22:00:00 2014(1406635200), 352076, 0, succeeded
[root@orilla ~]#
```

Figure 54: Listing savegroup completion information for a specific group

To access the details of a prior savegroup, use the *-t timestamp* option and specify either the nsavetime or the exact time (down to the second) of the group execution:

```
# nsrsgrpcmp -t timestamp groupName
```

For example:

```
[root@orilla ~]# nsrsgrpcmp -t "Sun Jul 27 22:00:00 2014" "Synology Saturday"
orilla.turbamentis.int, savefs, "succeeded:full:savefs"
* orilla.turbamentis.int:savefs savefs orilla.turbamentis.int: succeeded.
orilla.turbamentis.int, /synology/finance, "succeeded:incr:save"
* orilla.turbamentis.int:/synology/finance 86705:save: Successfully established
DFA session with adv_file device for save-set ID '3772049876' (orilla.turbamenti
s.int:/synology/finance).
* orilla.turbamentis.int:/synology/finance orilla.turbamentis.int: /synology/fin
ance level=incr, 167 KB 00:00:01 10 files
```

Figure 55: Accessing details of a specific savegroup execution with `nsrsgrpcmp`

Note that groups are only accessible via this command for the period they are retained in the jobs database.

You can equally get a list of *all* groups whose completion details can be printed by omitting the group name in the `list` command:

```
# nsrsgrpcmp -L
```

```
[root@orilla ~]# nsrsgrpcmp -L
name, start time, job id, previous job id, completion status
Slow Servers, Sun Jul 27 21:00:01 2014(1406458801), 320509, 0, failed
Laptops, Sun Jul 27 21:35:00 2014(1406460900), 320519, 0, failed
Synology Saturday, Sun Jul 27 22:00:00 2014(1406462400), 320528, 0, succeeded
Hyperion non-iTunes, Sun Jul 27 22:45:01 2014(1406465101), 320537, 0, succeeded
Hyperion iTunes, Sun Jul 27 23:55:00 2014(1406469300), 320542, 0, succeeded
Slow NMC, Mon Jul 28 06:00:00 2014(1406491200), 320547, 0, succeeded
Slow Servers, Mon Jul 28 21:00:00 2014(1406545200), 352001, 0, failed
Laptops, Mon Jul 28 21:35:01 2014(1406547301), 352014, 0, failed
Synology Saturday, Mon Jul 28 22:00:00 2014(1406548800), 352026, 0, succeeded
Hyperion non-iTunes, Mon Jul 28 22:45:01 2014(1406551501), 352034, 0, succeeded
Hyperion iTunes, Mon Jul 28 23:55:01 2014(1406555701), 352039, 0, succeeded
Slow NMC, Tue Jul 29 06:00:00 2014(1406577600), 352043, 0, succeeded
Slow Servers, Tue Jul 29 21:00:00 2014(1406631600), 352052, 0, succeeded
Laptops, Tue Jul 29 21:35:00 2014(1406633700), 352065, 0, failed
Synology Saturday, Tue Jul 29 22:00:00 2014(1406635200), 352076, 0, succeeded
Hyperion non-iTunes, Tue Jul 29 22:45:00 2014(1406637900), 352085, 0, succeeded
Hyperion iTunes, Tue Jul 29 23:55:00 2014(1406642100), 352090, 0, succeeded
Slow NMC, Wed Jul 30 06:00:01 2014(1406664001), 352094, 0, succeeded
[root@orilla ~]#
```

Figure 56: Obtaining a list of all available savegroup completion reports

To restrict the report to a single client, use the command:

```
# nsrsgrpcmp -c clientName [-t timestamp] groupName
```

For an administrator debugging an issue at the command line this makes for a good way of quickly checking what the output from the group was, even without access to email:

```
[root@orilla ~]# nsrgrpcmp -c miranda -t "Sun Jul 27 21:00:01" "Slow Servers"
miranda, savefs, "succeeded:full:savefs"
* miranda:savefs savefs miranda: succeeded.
miranda, /, "failed:full:save"
* miranda:/ miranda://: retried 1 times.
* miranda:/ Warning: `/Library/Caches/CrashPlan/61/cpbfbf000000000000000000000000/cpbmf'
' changed during save
* miranda:/ Warning: `/Library/Caches/CrashPlan/61/cpfmf' changed during save
* miranda:/ Warning: `/Library/Caches/CrashPlan/61/cphdf' size grew during save
* miranda:/ Warning: `/Library/Logs/CrashPlan/backup_files.log.0' size grew duri
ng save
* miranda:/ Warning: `/Library/Logs/ProfileManager/xpostgres.log' size grew duri
ng save
* miranda:/ Warning: `/Library/Server/Wiki/Database.xpg/backup/000000010000000000
000004B.partial' changed during save
* miranda:/ Warning: `/Library/Server/Wiki/Logs/postgres-xpg.log' size grew duri
ng save
```

Figure 57: Accessing details of a specific client and group

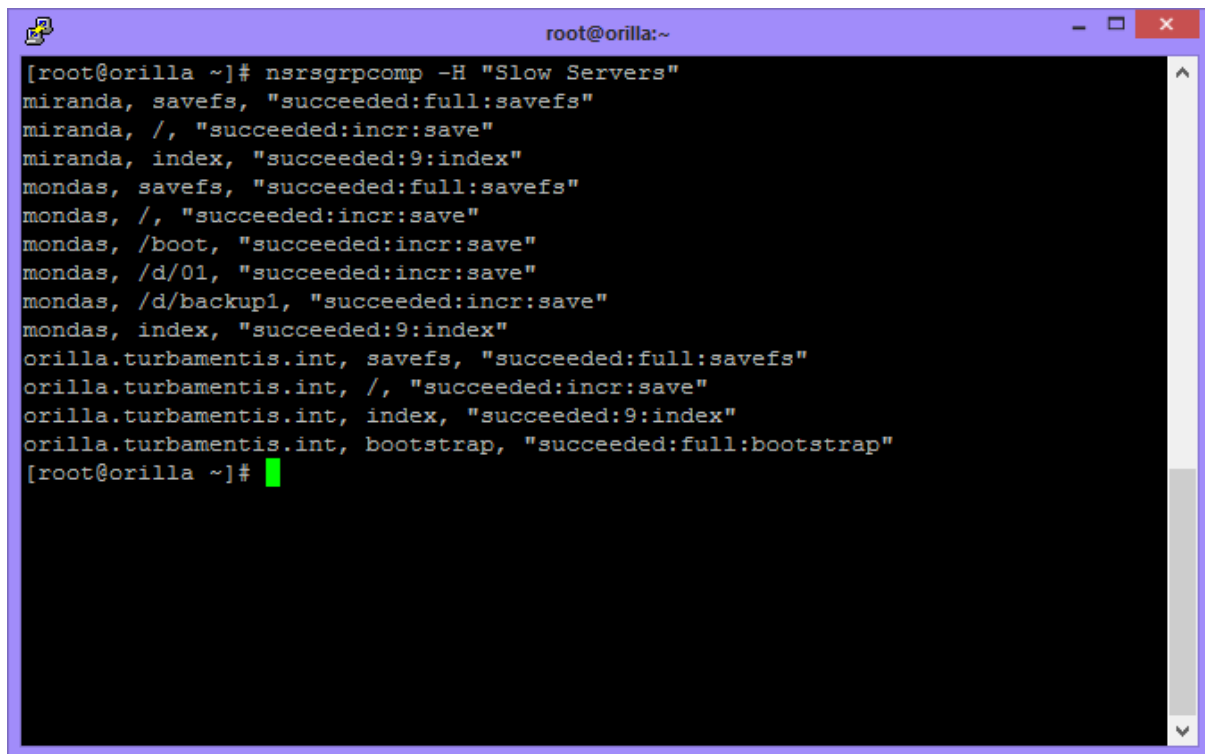
An astute NetWorker administrator will know that these reports are also stored in the *nsr/logs/sq* directory, with each group being a named directory underneath. However, the advantage of *nsrsrgrpcomp* is that it allows for information retrieval without searching those files, *and* the information may be retrieved from a host other than the backup server using the “-s server” option:

```
[Wed Jul 30 07:08:24]
pmdg@hyperion ~
$ nrsgrpcmp -s orilla -c miranda -t "Sun Jul 27 21:00:01" "Slow Servers"
miranda, savefs, "succeeded:full:savefs"
* miranda:savefs savefs miranda: succeeded.
miranda, /, "failed:full:save"
* miranda:/ miranda:/: retried 1 times.
* miranda:/ Warning: `/Library/Caches/CrashPlan/61/cpbfbf0000000000000000000000000000/cpbmf'
' changed during save
* miranda:/ Warning: `/Library/Caches/CrashPlan/61/cpfmfbf' changed during save
* miranda:/ Warning: `/Library/Caches/CrashPlan/61/cphdfbf' size grew during save
* miranda:/ Warning: `/Library/Logs/CrashPlan/backup_files.log.0' size grew duri
ng save
* miranda:/ Warning: `/Library/Logs/ProfileManager/xpostgres.log' size grew duri
ng save
* miranda:/ Warning: `/Library/Server/Wiki/Database.xpg/backup/0000000010000000000
0000004B.partial' changed during save
```

Figure 58: Running nsrgrpcomp from a host other than the NetWorker server

Another advantage of `nrsgrpcomp` is the option to extract a subset of the information relating to a backup. For instance, summary information may be extracted by using the `-H` option:

```
# nsrsgroupcomp -H "Slow Servers"
```

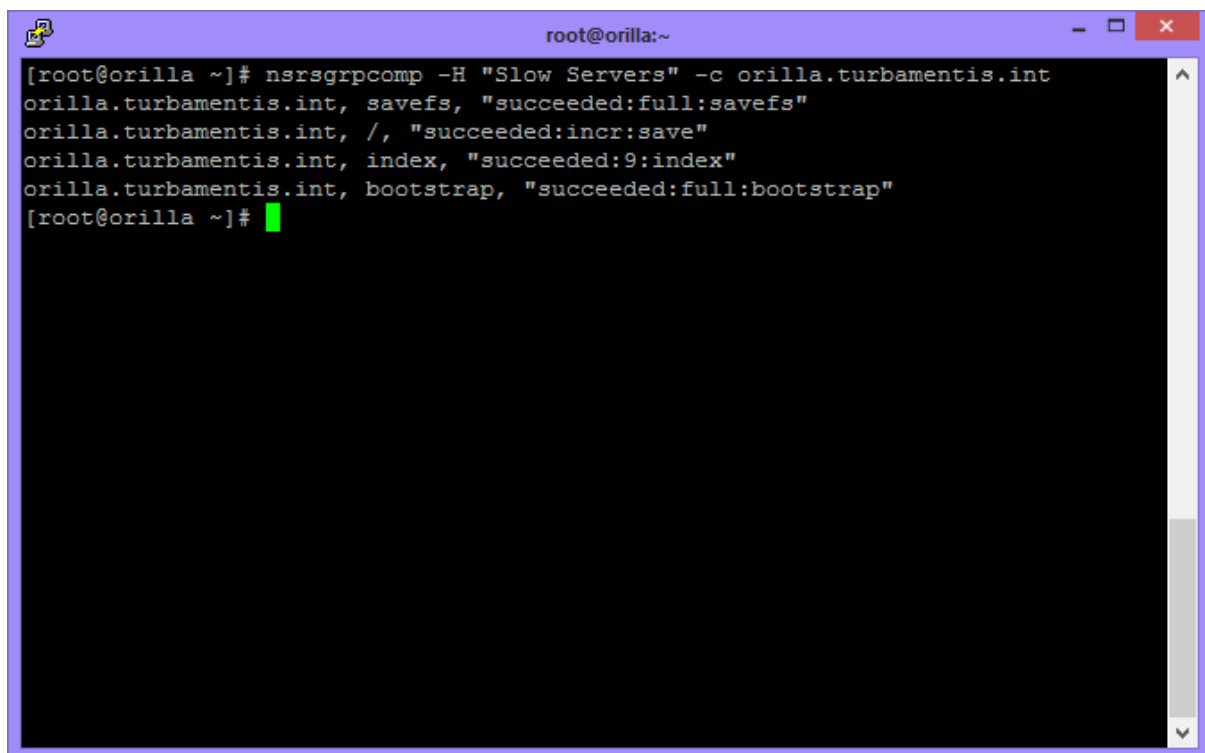


```
root@orilla:~  
[root@orilla ~]# nsrsgrpcmp -H "Slow Servers"  
miranda, savefs, "succeeded:full:savefs"  
miranda, /, "succeeded:incr:save"  
miranda, index, "succeeded:9:index"  
mondas, savefs, "succeeded:full:savefs"  
mondas, /, "succeeded:incr:save"  
mondas, /boot, "succeeded:incr:save"  
mondas, /d/01, "succeeded:incr:save"  
mondas, /d/backup1, "succeeded:incr:save"  
mondas, index, "succeeded:9:index"  
orilla.turbamentis.int, savefs, "succeeded:full:savefs"  
orilla.turbamentis.int, /, "succeeded:incr:save"  
orilla.turbamentis.int, index, "succeeded:9:index"  
orilla.turbamentis.int, bootstrap, "succeeded:full:bootstrap"  
[root@orilla ~]#
```

Figure 59: Extracting summary information from nsrsgrpcmp

The summary information may be further refined by specifying a single client:

```
# nsrsgrpcmp -H "Slow Servers" -c orilla.turbamentis.int
```



```
root@orilla:~  
[root@orilla ~]# nsrsgrpcmp -H "Slow Servers" -c orilla.turbamentis.int  
orilla.turbamentis.int, savefs, "succeeded:full:savefs"  
orilla.turbamentis.int, /, "succeeded:incr:save"  
orilla.turbamentis.int, index, "succeeded:9:index"  
orilla.turbamentis.int, bootstrap, "succeeded:full:bootstrap"  
[root@orilla ~]#
```

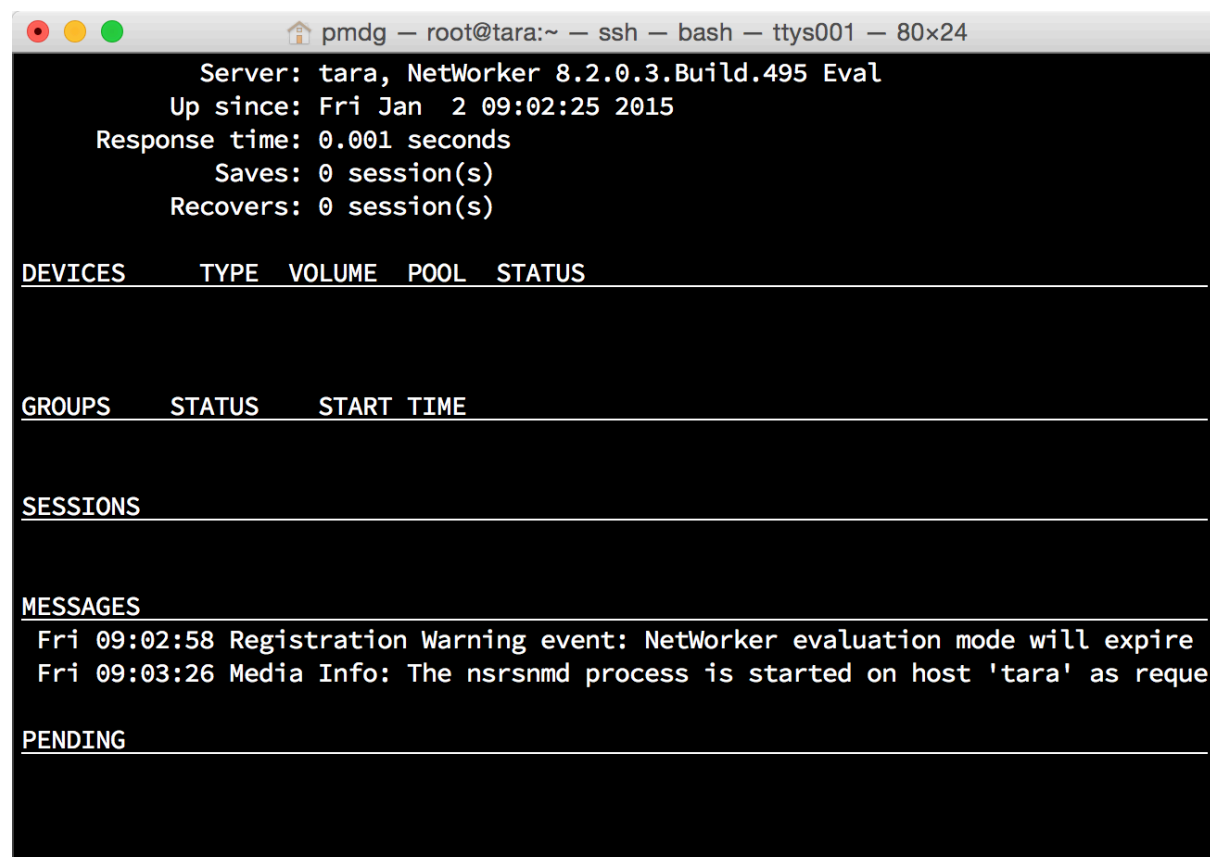
Figure 60: Refining nsrsgrpcmp summary output, by client

11.3 Using nsrwatch

The nsrwatch utility has been a mainstay of NetWorker on Linux and Unix platforms for a *very* long time. (I remember using it in NetWorker v4.x, the oldest version I used as a server.)

In the most basic form, you can invoke it just by running:

```
# nsrwatch
```



```
Server: tara, NetWorker 8.2.0.3.Build.495 Eval
Up since: Fri Jan  2 09:02:25 2015
Response time: 0.001 seconds
Saves: 0 session(s)
Recovers: 0 session(s)

DEVICES  TYPE  VOLUME  POOL  STATUS

GROUPS   STATUS  START TIME

SESSIONS

MESSAGES
Fri 09:02:58 Registration Warning event: NetWorker evaluation mode will expire
Fri 09:03:26 Media Info: The nsrsnmd process is started on host 'tara' as reque

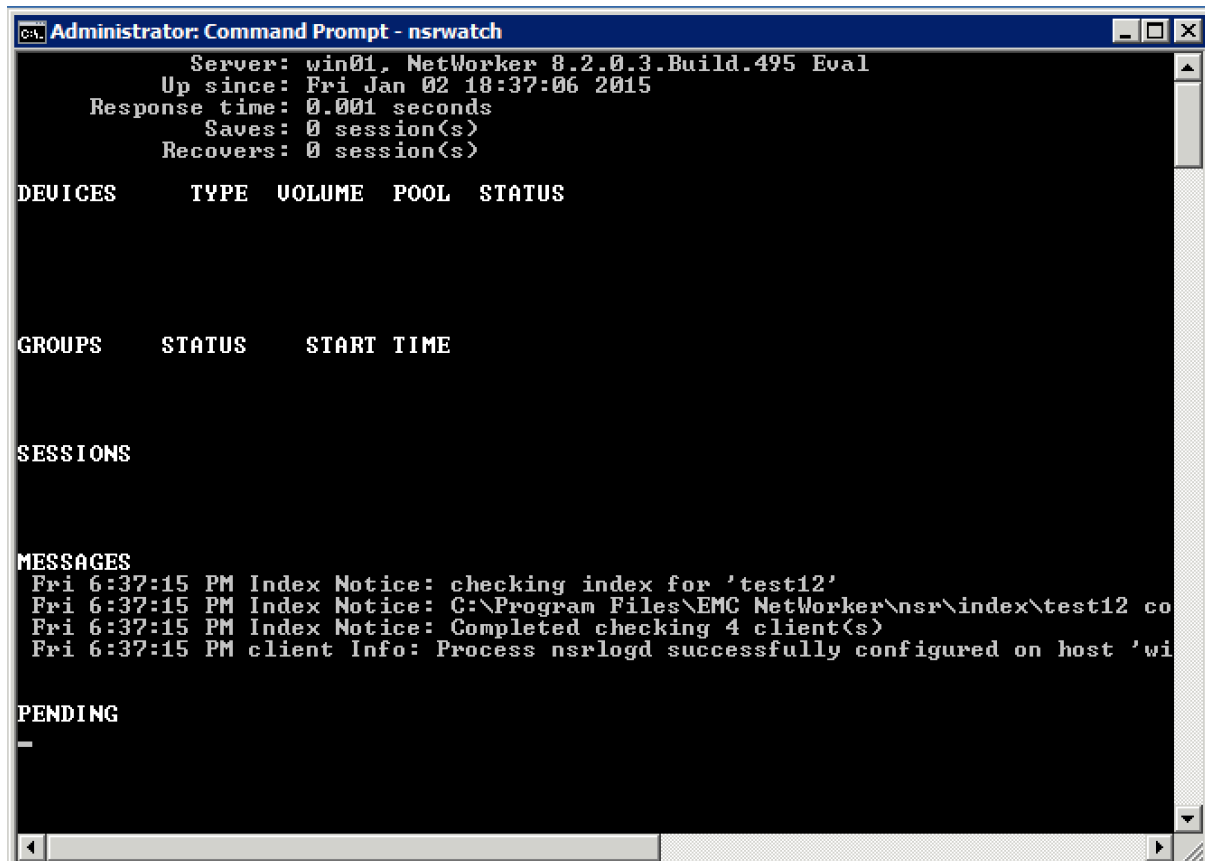
PENDING
```

Figure 61: nsrwatch on a Unix platform

The nsrwatch screen is broken up into several key areas:

- **Summary information** – Server name, how long server has been running for, number of saves and recoveries, etc.
- **Devices** – Each device on the server, what volume/pool is mounted on the device, and the status (messages) for the device.
- **Groups** – Actively executing groups (and sometimes recently completed groups)
- **Sessions** – What backup, recovery or clone sessions are currently active.
- **Messages** – Key messages output by the backup server
- **Pending** – Notifications of where the NetWorker server requires intervention by an administrator or operator.

Gone are the days when nsrwatch was only available for the Unix platform, however. On any modern NetWorker environment, you'll also find nsrwatch to be part of the Windows client package as well.



```
Administrator: Command Prompt - nsrwatch
Server: win01, NetWorker 8.2.0.3.Build.495 Eval
Up since: Fri Jan 02 18:37:06 2015
Response time: 0.001 seconds
Saves: 0 session(s)
Recovers: 0 session(s)

DEVICES      TYPE  VOLUME  POOL  STATUS

GROUPS      STATUS  START TIME

SESSIONS

MESSAGES
Fri 6:37:15 PM Index Notice: checking index for 'test12'
Fri 6:37:15 PM Index Notice: C:\Program Files\EMC NetWorker\nsr\index\test12 co
Fri 6:37:15 PM Index Notice: Completed checking 4 client(s)
Fri 6:37:15 PM client Info: Process nsrlogd successfully configured on host 'wi

PENDING
-
```

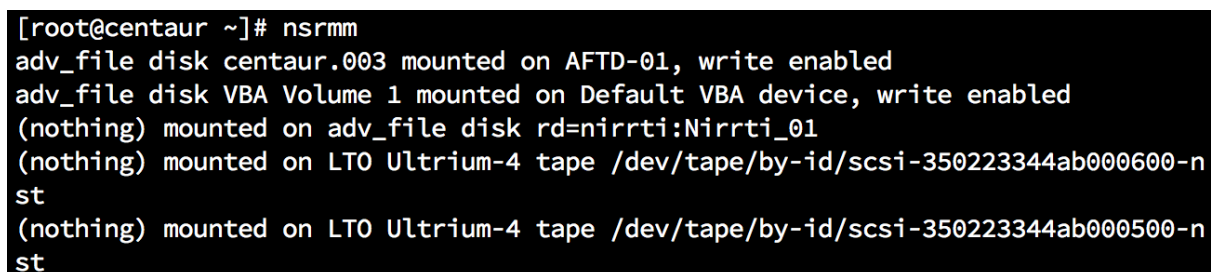
Figure 62: Running nsrwatch on Windows

You'll note that I mentioned nsrwatch is part of the Windows client package on modern NetWorker installs. Similarly, it has always been part of the *client* package for Unix and Linux installs. This immediately gives us our first option for running nsrwatch:

```
# nsrwatch -s server
```

This will launch nsrwatch, pointing it at the nominated NetWorker server. Note that the results of this may be variable if the host you're running nsrwatch from is not a defined client of the NetWorker server you're intending to watch.

One of the things you'll immediately note with nsrwatch is the amount you can see on screen is directly proportional to your window size. For instance, consider a server with multiple devices:



```
[root@centaur ~]# nsrmm
adv_file disk centaur.003 mounted on AFTD-01, write enabled
adv_file disk VBA Volume 1 mounted on Default VBA device, write enabled
(nothing) mounted on adv_file disk rd=nirrti:Nirrti_01
(nothing) mounted on LT0 Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000600-nst
(nothing) mounted on LT0 Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000500-nst
```

Figure 63: More devices than nsrwatch will show

On such a server, running nsrwatch won't immediately yield very much information about the devices:

DEVICES	TYPE	VOLUME
/dev/tape/by-id/scsi-350223344ab000100-nst	(J) LTO Ultrium-4	
/dev/tape/by-id/scsi-350223344ab000200-nst	(J) LTO Ultrium-4	
GROUPS	STATUS	START TIME
SESSIONS		
MESSAGES		
Fri 06:45:37 PM Media Info: Jukebox 'VTL1' Hardware status of jukebox 'VTL1' is		
Fri 06:46:03 PM Media Info: The storage node centaur is ready for use.		
PENDING		

Figure 64: nsrwatch with limited screen space

However, that doesn't mean you can't see anything about the other devices. You can use the tab key to jump between sections of display in nsrwatch, then when you're within a highlighted section, use the up and down arrow keys to scroll through the additional information in that section:

DEVICES	TYPE	VOLUME
/dev/tape/by-id/scsi-350223344ab000600-nst	(J) LTO Ultrium-4	
AFTD-01	adv_file	centaur.003
GROUPS	STATUS	START TIME
SESSIONS		
MESSAGES		
Fri 06:45:37 PM Media Info: Jukebox 'VTL1' Hardware status of jukebox 'VTL1' is		
Fri 06:46:03 PM Media Info: The storage node centaur is ready for use.		
PENDING		

Figure 65: Tabbing between different sections of nsrwatch

When in nsrwatch, you can turn off or on various sections by tapping particular keys:

Table 3: Display toggle options in nsrwatch

Key	Display Field Toggled
d/D	Devices. Optional: a second tap of 'd' will display only <i>mounted</i> devices ¹¹ .
g/G	Groups
h/H	Help
j/J	Jobs

¹¹ If the devices are already shown – such as when you first launch nsrwatch, the first tap will switch to only mounted devices.

Key	Display Field Toggled
m/M	Messages
p/P	Pending alerts
s/S	Active sessions
t/T	Tunnel connections
y/Y	VMware protection policy

These allow you to quickly configure nsrwatch to just show you the information you're specifically after in a viewing session. For instance, the following shows the option to only display mounted devices:

DEVICES (mounted)		TYPE	VOLUME
AFTD-01		adv_file	centaur.003
Default VBA device		adv_file	VBA Volume
GROUPS	STATUS	START TIME	
SESSIONS			
MESSAGES			
Fri 06:45:37 PM Media Info: Jukebox `VTL1' Hardware status of jukebox 'VTL1' is			
Fri 06:46:03 PM Media Info: The storage node centaur is ready for use.			

Figure 66: Viewing only mounted devices in nsrwatch

A final option to consider when using nsrwatch is the *polling interval* option. If your NetWorker server is very busy (e.g., hundreds of simultaneous savesets backing up), or you're connecting to it via a very slow link, you may want to change the interval between polls of the NetWorker server. This will slow down the updates in nsrwatch, but will be less impactful to a busy server or less bandwidth intensive over a slow link. The default poll time is two seconds, but can be changed by using a '-p seconds' option when invoking nsradmin. For example:

```
# nsrwatch -p 60
```

The above command would invoke nsrwatch with a refresh ratio of once every minute.

Control Commands

12 Introduction

Many of the NetWorker command line utilities are to do with *control* of the application. Regardless of whether you want to do something with backup and recovery, the media database or the configuration, it's likely there is a command line utility to help you.

We're not going to look at *all* the command line options for NetWorker. Indeed, in particular, backup and recovery options (for the most part) will be ignored – these are very well documented in the respective guides, and don't need additional coverage here.

Key topics we'll be examining are:

- Devices and the Media Database
- Configuration
- Tape Libraries

13 nsrmm

13.1 What is nsrmm?

The *nsrmm* utility is your media management command. The activities you can use it for are neatly bracketed into two distinct categories:

- Media manipulation
- Media database manipulation

13.2 Warning

This topic describes activities that, if misused, could cause loss of data on a NetWorker server, either through removal of media database/saveset records, or through the premature overwriting of NetWorker media.

Extreme caution should be taken with this topic, and the commands described should only be run on a lab server unless you are absolutely certain with what you are doing and have practiced the command in a lab first.

13.3 Lab Environment

For the exercises involving potentially destructive operations, we will use a lab server that has two disk backup devices defined on it, *Disk1* and *Disk2*. In addition to the standard bootstrap pools automatically created by NetWorker, we will have two additional pools:

- Backup pool: DiskBackup
- Backup Clone pool: DiskClone

It is recommended you install NetWorker 8 or higher on the lab server – NetWorker 7.x and lower uses slightly different disk device identification methods than v8.

13.4 Media manipulation

If you run *nsrmm* without any arguments, it will print the current volume status of all devices:

```
# nsrmm
```

```
[root@orilla ~]# nsrmm
adv_file disk Staging.02 mounted on rd=mondas:Standard.02, write enabled
adv_file disk Slow.01 mounted on Slow-01, write enabled
adv_file disk Standard.01 mounted on Standard-01, write enabled
[root@orilla ~]#
```

Figure 67: Default execution of *nsrmm*

Note that *nsrmm* returns the status of *all* devices, including tape (physical or virtual). For instance:

```
# nsrmm
```

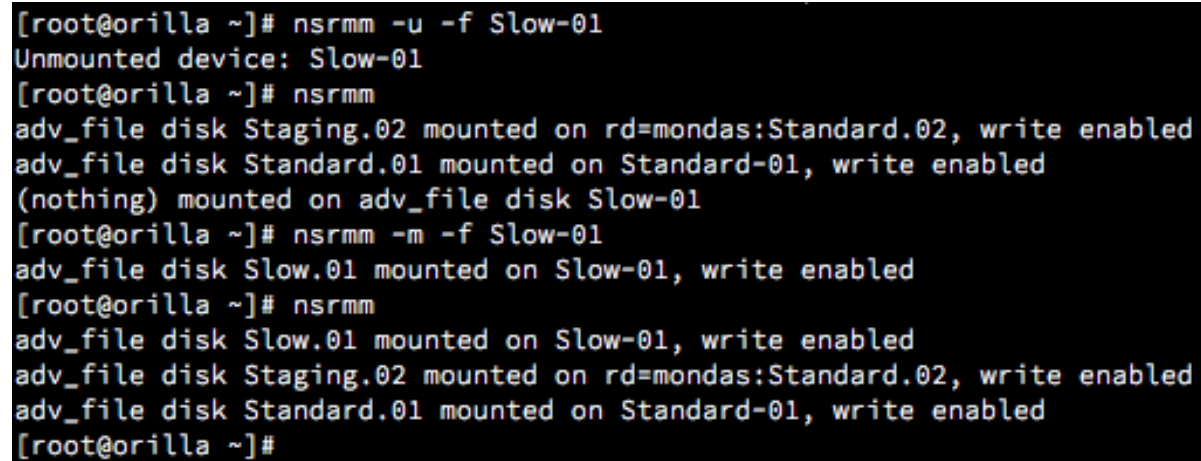
```
[root@tara ~]# nsrmm
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000500-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000400-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000600-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000300-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000200-nst
(nothing) mounted on adv_file disk Disk1
(nothing) mounted on adv_file disk Disk2
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000100-nst
[root@tara ~]#
```

Figure 68: *nsrmm* output showing combined tape/disk device status

nsrmm may be used to mount or unmount volumes from standalone devices¹² with the following respective commands:

```
# nsrmm -m -f device
# nsrmm -u -f device
```

For instance:



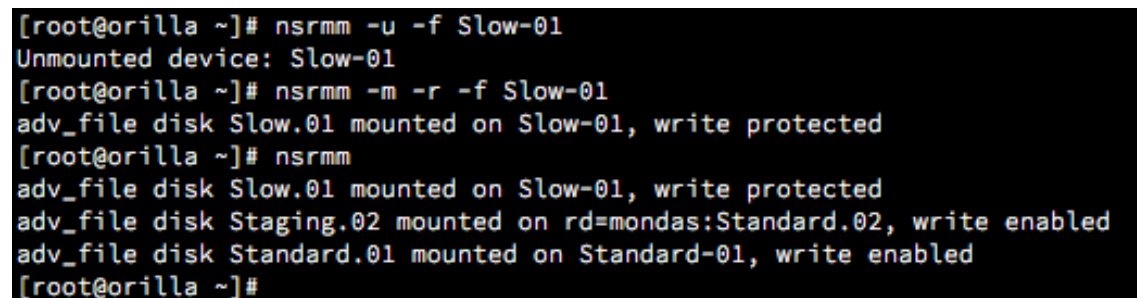
```
[root@orilla ~]# nsrmm -u -f Slow-01
Unmounted device: Slow-01
[root@orilla ~]# nsrmm
adv_file disk Staging.02 mounted on rd=mondas:Standard.02, write enabled
adv_file disk Standard.01 mounted on Standard-01, write enabled
(nothing) mounted on adv_file disk Slow-01
[root@orilla ~]# nsrmm -m -f Slow-01
adv_file disk Slow.01 mounted on Slow-01, write enabled
[root@orilla ~]# nsrmm
adv_file disk Slow.01 mounted on Slow-01, write enabled
adv_file disk Staging.02 mounted on rd=mondas:Standard.02, write enabled
adv_file disk Standard.01 mounted on Standard-01, write enabled
[root@orilla ~]#
```

Figure 69: Unmounting and mounting volumes with nsrmm

Equally, a device can be mounted *read-only* via nsrmm using the following command:

```
# nsrmm -m -r -f device
```

For instance:



```
[root@orilla ~]# nsrmm -u -f Slow-01
Unmounted device: Slow-01
[root@orilla ~]# nsrmm -m -r -f Slow-01
adv_file disk Slow.01 mounted on Slow-01, write protected
[root@orilla ~]# nsrmm
adv_file disk Slow.01 mounted on Slow-01, write protected
adv_file disk Staging.02 mounted on rd=mondas:Standard.02, write enabled
adv_file disk Standard.01 mounted on Standard-01, write enabled
[root@orilla ~]#
```

Figure 70: Mounting a volume in read-only mode with nsrmm

Many of the volume manipulation commands (via nsrmm for media database interaction) will require volumes to be unmounted.

Note if you've mounted a volume as read-only, you need to unmount the volume in order to *remount* it as read-write:

¹² Actually, it can also be used in certain circumstances on jukebox devices as well, but it's generally not recommended.

```
[root@orilla ~]# nsrmm -m -f Slow-01
6183:nsrmm: Slow-01 is already mounted
[root@orilla ~]# nsrmm -u -f Slow-01
Unmounted device: Slow-01
[root@orilla ~]# nsrmm -m -f Slow-01
adv_file disk Slow.01 mounted on Slow-01, write enabled
[root@orilla ~]#
```

Figure 71: 'Remounting' a read-only volume as read-write

The reason for this requirement is hinted at in the previous text: some changes to volume state can only be achieved when the volume is not mounted – and the transition to read-only from read-write or vice-versa is one of these.

The nsrmm utility can also be used to label a volume. The command for this is:

```
# nsrmm -m -l -b pool -f device volName
```

Where:

- **pool** is the name of the pool the volume is to be assigned to; this must be specified in *exactly the same case* as it is defined in NetWorker. If the pool name has spaces in it, enclose it in quotes.
- **device** is the name of the device where the volume is (if tape) or is to be defined (if disk).
- **volName** is the intended name of the volume.

CAUTION - Lab Exercise

To label a volume on the device *Disk1* into the *DiskBackup* pool, with a volume label of *Disk.01*, the command used would be:

```
# nsrmm -m -l -b DiskBackup -f Disk1 Disk.01
```

The default nsrmm command can be run after this to verify the completed operation:

```
[root@tara ~]# nsrmm -m -l -b DiskBackup -f Disk1 Disk.01
[root@tara ~]# nsrmm
adv_file disk Disk.01 mounted on Disk1, write enabled
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000500-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000400-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000600-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000300-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000200-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000100-nst
(nothing) mounted on adv_file disk Disk2
[root@tara ~]#
```

Figure 72: Labelling a volume using nsrmm

Note that the '-m' option in this command is not, technically required. However, running the command without the '-m' option will result in a label-without-mount operation, which is *usually* undesirable. We can test this with the second disk backup unit:

```
# nsrmm -l -b DiskBackup -f Disk2 Disk02
```

```
[root@tara ~]# nsrmm -l -b DiskBackup -f Disk2 Disk.02
[root@tara ~]# nsrmm
(nothing) mounted on adv_file disk Disk2
adv_file disk Disk.01 mounted on Disk1, write enabled
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000500-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000400-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000600-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000300-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000200-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000100-nst
[root@tara ~]#
[root@tara ~]# nsrmm -m -f Disk2
adv_file disk Disk.02 mounted on Disk2, write enabled
[root@tara ~]#
```

Figure 73: Labelling a volume without mounting it using nsrmm

(The label-without-mount operation might be used in scenarios where a volume needs to be labelled, but you don't want it used *yet*.)

A volume in NetWorker may be *relabelled* by using the label-with-relabel option:

```
# nsrmm -mlR -f device
```

You'll notice in the above command that we're *stacking* arguments. This is a good short cut to get used to in NetWorker. That is, instead of using arguments '-m -l -R', they've been reduced to '-mlR'. Note as well that in theory the entire command could be reduced to '-mlRf device' – however, it's generally advisable to maintain a habit of only stacking command line arguments that don't take options.

CAUTION - Lab Exercise

```
[root@tara ~]# nsrmm -mlR -f Disk2
Volume 'Disk.02' is not recyclable - are you sure you want to over-write it with a new label?
y
[root@tara ~]#
```

Figure 74: Relabeling a volume with nsrmm

If the volume you're attempting to relabel is not actually recyclable, you'll be prompted to confirm whether you really want to proceed. This can be overridden by including a '-y' flag, but you should *apply extreme caution* to including the '-y' option in nsrmm commands. Only do so when you are absolutely certain what you want to do:

```
[root@tara ~]# nsrmm -mlRy -f Disk2
[root@tara ~]#
```

Figure 75: Automatically answering 'yes' to nsrmm prompts (dangerous)

The warning provided by NetWorker regarding a volume relabel is very real:

- For tapes, NetWorker will write a new label header and new double-EOF to the media; NetWorker will never seek beyond the media – recovery from an erroneously labelled tape will require forensic intervention by a specialist data-recovery company
- For file, adv_file and Boost devices, NetWorker will immediately *delete* all the stored savesets on the filesystem

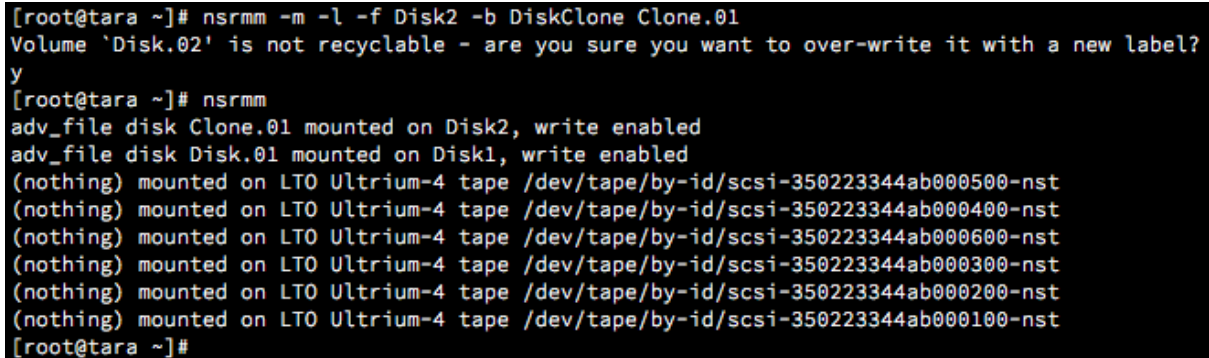
There are actually two styles of relabel actions against volumes. The one we've used so far is for scenarios where you want to simply overwrite the existing volume but retain its label and pool.

Consider the scenario however where the goal is to relabel a volume mistakenly placed into the incorrect pool. For instance, with two disk devices of Disk1 and Disk2, it might have been desirable to label the volume for Disk2 into the DiskClone pool instead.

CAUTION - Lab Exercise

In this scenario, the relabeling operation reverts to a standard label operation instead:

```
# nsrmm -m -l -f Disk2 -b DiskClone Clone.01
```



```
[root@tara ~]# nsrmm -m -l -f Disk2 -b DiskClone Clone.01
Volume 'Disk.02' is not recyclable - are you sure you want to over-write it with a new label?
y
[root@tara ~]# nsrmm
adv_file disk Clone.01 mounted on Disk2, write enabled
adv_file disk Disk.01 mounted on Disk1, write enabled
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000500-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000400-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000600-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000300-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000200-nst
(nothing) mounted on LTO Ultrium-4 tape /dev/tape/by-id/scsi-350223344ab000100-nst
[root@tara ~]#
```

Figure 76: Labelling a previously labelled volume into a new pool

13.5 Media Database manipulation

While technically labelling or relabeling volumes is particular form of media database manipulation, nsrmm can be used to more precisely affect the state of the media database by targeting either savesets or volumes for specific operations.

There are several particularly useful options for nsrmm when it comes to media database manipulation. These are:

- Changing the mode of a volume or saveset (-o *mode*)
- Deleting a volume or saveset (-d)
- Changing the browse and/or retention time of a saveset (-w *time* and -e *time* respectively)
- Erasing an advanced file type or Boost device (-E).

All of these options should be used with caution.

13.6 Changing the mode of a volume or saveset

You can change the *mode* of a volume or saveset by invoking nsrmm as follows:

```
# nsrmm -o mode {-S SSID[/CID] | -V volumeID | VolumeName}
```

Where:

- '*mode*' is one of the following:
 - recyclable – Flags a saveset, saveset clone or volume recyclable
 - notrecyclable – Flags a saveset, saveset clone or volume recyclable
 - readonly – Flags a volume read-only
 - notreadonly – Flags a volume not read-only
 - offsite – Flags a volume as being offsite (quite distinct from setting an 'offsite' location in *mmlocate*)
 - notoffsite – Clears the offsite flag for a volume
 - scan – Flags a volume as requiring scanning
 - notscan – Flags a volume as not requiring scanning
 - full – Flags a volume as being full

- notfull – Flags a volume as being appendable¹³ (i.e., not full)
- manual – Flags a volume as requiring manual recycling (will not be automatically recycled)
- notmanual – Flags a volume as not requiring manual recycling (can be automatically recycled)
- suspect – Flags a volume as suspect (not eligible for consideration for recoveries)
- notsuspect – Removes the suspect flag for a volume
- *SSID* is a specific saveset ID to run the command against, or more specifically, *SSID/CID* is a specific saveset ID and specific instance of a *clone* of that saveset to run the command against
- *volumeID* is a specific volume ID to run the command against
- *volumeName* is a specific volume name (label) to run the command against.

For sites still using physical tape, and particularly a mix of physical tape and disk backup devices, the *offsite/notoffsite* flags are particularly useful. Consider the eligibility criteria NetWorker considers volumes for in order of recovery priority when the same saveset has multiple copies:

- Volumes currently mounted will receive highest priority
- Volumes that are *nearline* will receive next highest priority (i.e., in a tape library)
- If no volume is mounted or nearline, NetWorker will request the volume that has the saveset instance with the *lowest* clone ID.

It's the third option that isn't always desirable. If using traditional disk backup devices (i.e., ADV_FILE devices) as staging locations, a reasonably typical process is:

- Backup to disk
- Clone immediately to tape, sent off-site
- (Later) Stage from disk to tape, removing disk copy. Tape copy is stored locally.

This model uses disk for short-term storage, with regular or longer-term retention periods being serviced by tape. Since there should never be a single instance of any backup, savesets are copied to tape twice – the first time as an actual clone operation, and the second as a staging operation (usually when the disk backup devices have reached a particular capacity, or the savesets have reached a particular age).

Each time NetWorker generates a copy of a saveset (including the original, first copy), it tags the saveset instance as having a particular *clone ID*. This clone ID is the time the instance was generated expressed as seconds since January 1, 1970 (GMT). Even if a saveset is *staged*, a new instance is created (with a new clone ID), and the old is destroyed.

Thus, in our above scenario:

- Backup to disk (saveset with clone ID X)
- Clone to tape (saveset with clone ID Y, where Y>X)
- Stage to tape (saveset with clone ID Z, where Z>Y).

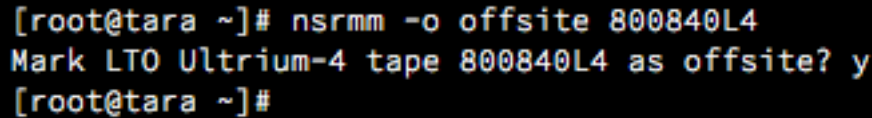
Therefore, if NetWorker needs to read from that particular saveset and all copies are currently on tape, but neither tape is in a library, it will request the *clone* volume since it has the copy with the *lowest* clone ID (the original instance, with clone ID X, has been removed from disk and the media database by now).

¹³ Of course, marking a volume 'notfull' that NetWorker has flagged as full because it encountered the end of physical media (or filled the device) will not allow additional writes to happen to the end of the volume. It will just waste time.

If the neither volume is in the library but you want NetWorker to target the recovery from the staged copy, which presumably *is* on-site, you can use the command:

```
# nsrmm -o offsite cloneVolume
```

To tag the clone volume with the label identified by *cloneVolume* as being offsite.



```
[root@tara ~]# nsrmm -o offsite 800840L4
Mark LTO Ultrium-4 tape 800840L4 as offsite? y
[root@tara ~]#
```

Figure 77: Flagging a volume as being offsite

Thus, we can say that NetWorker actually uses the following method for prioritising volumes for recovery purposes:

- Volumes currently mounted will receive highest priority
- Volumes that are *nearline* will receive next highest priority (i.e., in a tape library)
- If no volume is mounted or nearline, NetWorker will request the volume that has the saveset instance with the *lowest* clone ID which is *not* flagged as being offsite.

(A good tip if you're using the 'backup to disk, clone to tape, stage to tape' workflow is to have operators or administrators automatically flag clone volumes as being *offsite* as they are sent offsite.)

Two of the other common mode settings with nsrmm is to flag volumes as read-only and to change the recyclability status of either a volume or saveset.

```
# nsrmm -o readonly volumeName
```

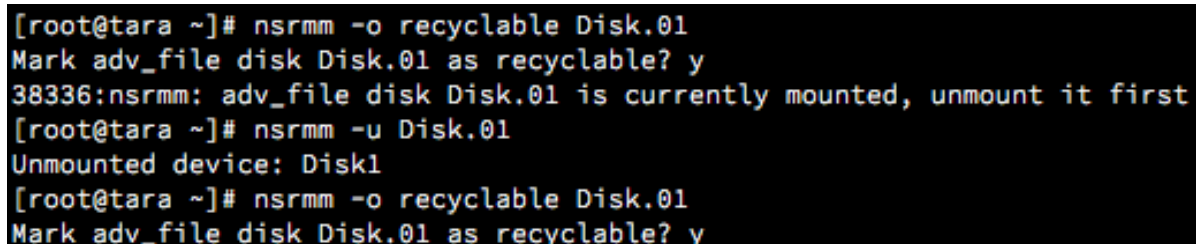
Will mark the volume identified by '*volumeName*' as being readonly. Note that this **does not** protect the volume from being recycled, merely being written to, so it should be used with caution.

When used against a volume, the -o recyclable option will flag the entire volume as being recyclable:

```
# nsrmm -o recyclable volumeName
```

This not only flags the *volume* as recyclable, but *all* savesets on the volume as being recyclable. Thus, for disk (ADV_FILE or Boost) devices, the next media cleaning/consistency check operation (nsrim -X) will trigger a purge of the savesets from disk (if it has been subsequently mounted).

Consider the following scenario where a backup was written to the Disk.01 volume and it was subsequently marked as recyclable:



```
[root@tara ~]# nsrmm -o recyclable Disk.01
Mark adv_file disk Disk.01 as recyclable? y
38336:nsrmm: adv_file disk Disk.01 is currently mounted, unmount it first
[root@tara ~]# nsrmm -u Disk.01
Unmounted device: Disk1
[root@tara ~]# nsrmm -o recyclable Disk.01
Mark adv_file disk Disk.01 as recyclable? y
```

Figure 78: Marking a volume as recyclable

Note in the example above that an attempt was made to mark the volume as recyclable while it was still mounted – NetWorker will not permit this to happen. Instead, the user is prompted to unmount the volume first.

In the example, we've also used a slightly different unmount command – unmounting by volume *name* rather than device name.

Once the volume has been marked as recyclable, a custom mminfo command will confirm for us that the saveset just generated has in fact been flagged as recyclable:

```
[root@tara ~]# mminfo -q "volume=Disk.01" -r client,savetime,sumsize,level,name,ssflags
client      date      size  lvl name                      ssflags
tara.pmdg.lab 22/06/14 1419 MB manual /root          vEF
[root@tara ~]#
```

Figure 79: Saveset marked as recyclable via a volume being marked as recyclable

The recyclability option can be specified for a single saveset, or saveset instance as well, using the command:

```
# nsrmm -o recyclable -S SSID
```

or:

```
# nsrmm -o recyclable -S SSID/CID
```

Where:

- *SSID* is a specific saveset
- *SSID/CID* is a specific saveset *instance*.

Be particularly mindful of situations where NetWorker allows you to specify either the saveset ID or the saveset ID/clone ID. In these situations, it means that the command you use, if run against the saveset ID only, will be applied to *all* instances of the saveset.

Note that some instances of NetWorker may require you to specify the Saveset ID/Clone ID combo when flagging a saveset as *not* recyclable.

13.7 Changing Browse/Retention Time of a Saveset

The default browse/retention time for any new client in EMC NetWorker is a month, and year, respectively. A common enough scenario when creating a new client is for the administrator or operator to forget to set the appropriate browse and retention policy for the backups (at least, until it becomes habitual).

This can mean that backups are generated with longer retention times than desired, and:

- If backing up to tape (physical or virtual), the media may not become recyclable at the anticipated time due to these savesets;
- If backing up to disk (adv_file or Boost), the volume's capacity may be more difficult to manage than anticipated.

The nsrmm utility can be used to either shrink or extend the browse and/or retention period for a saveset. This comes with a few practical considerations however:

- You cannot make the browse time longer than the retention time
- You cannot extend the browse time if the browse time has already expired

For instance, consider the '/root' backup cited in the previous example:

```
[root@tara ~]# mminfo -q "name=/root,savetime>=yesterday" -r ssid,level,ssbrowse,ssretent
  ssid      lvl browse  retent
2493929846 manual 27/06/14 27/06/14
[root@tara ~]#
```

Figure 80: mminfo output showing browse and retention time for a saveset

In this scenario, the backup was a manual one, set to expire just 5 days following the time of the backup, and is typical of the sort of manual backup performed by system administrator to preserve a few files prior to making a change. If the administrator subsequently decided that she wanted to extend that browse and retention time for a month, she could use nsrmm as follows:

```
# nsrmm -S SSID[/CID] -w bTime -e eTime
```

Where:

- *SSID* is the saveset ID of a single saveset
- *SSID/CID* is the saveset ID/clone ID of a single instance of a saveset
- *bTime* is the new browse time to be applied to the saveset or saveset instance
- *eTime* is the new retention (expiration) time to be applied to the saveset or saveset instance

The browse time and expiration time specified may be in any format accepted by NetWorker, which means either a literal date expression (e.g., “27/07/2014”), or a fuzzy date format (e.g., “+1 month”). Note that both date formats have their caveats:

- The *output date format* for mminfo will reveal to you what short-form date format should be used in specifying a new browse or retention date (e.g., the above mminfo output shows a date format of DD/MM/YYYY);
- The ‘fuzzy’ date format must be preceded by a plus sign to indicate that it refers to a date *in the future*.

Thus, the command in this example might be:

```
# nsrmm -S 2493929846 -w "+1 month" -e "+1 month"
```

```
[root@tara ~]# nsrmm -S 2493929846 -w "+1 month" -e "+1 month"
If this saveset has dependents with longer browse or retention periods, the dependent savesets will not be browsable after this saveset's browse and expiration dates.
Are you sure you want to change the browse policy expiration to 'Wed Jul 23 07:31:46 2014' and the retention policy expiration to 'Wed Jul 23 07:31:46 2014' for this saveset? (yes/no) [no] y
[root@tara ~]#
```

Figure 81: Changing the browse and retention time for a saveset

14 Tape library operations

14.1 nsrjb

A close cousin to the nsrmm command is the *nsrjb* command. While nsrmm focuses on devices and the media database, nsrjb focuses on controlling jukeboxes (autochangers)¹⁴.

¹⁴ Some of the media database manipulation commands we’ve previously looked at, such as *-o mode* are available with nsrjb as well. However, I usually recommend reserving nsrjb just for library operations.

Obviously, if you're not using physical or virtual tape libraries, you should feel free to skip this topic. If you have or can install a Linux based Lab NetWorker server and want to practice with tape libraries, you can make use of the LinuxVTL project. For details on LinuxVTL, download the micromanual from the NetWorker Information Hub at:

<http://nsrd.info/micromanuals/download-linuxvtl.php>

CAUTION - Lab Exercises

All the exercises described in the *nsrjb* section should be performed first in a lab if you are not familiar with them. Exercises may reset the hardware state of a jukebox, label or relabel media, or make media unavailable. Proceed with this topic with **caution**.

If you only have one jukebox, *nsrjb* will provide a basic summary of the contents of your jukebox when run with no arguments:

```
# nsrjb
```

Assuming you have a jukebox defined in NetWorker, this will produce output along the lines of the following:

```
[root@centaur ~]# nsrjb
      1:      VTL1      [enabled]
There is only one enabled and configured jukebox: VTL1

Jukebox VTL1: (Ready to accept commands)
slot  volume          pool      barcode   volume id      recyclable
  1: 800001L4        VTL Backup 800001L4 3515159528      no
  2: 800002L4        VTL Backup 800002L4 3682931616      no
  3: 800003L4        VTL Backup 800003L4 3699708832      no
  4: 800004L4        VTL Backup 800004L4 3666154400      no
  5: 800005L4        VTL Backup 800005L4 3716486024      no
  6: 800006L4        VTL Backup 800006L4 3750040456      no
  7: 800007L4        VTL Backup 800007L4 3766817648      no
  8: 800008L4        VTL Backup 800008L4 3783594864      no
  9: 800009L4        VTL Backup 800009L4 3733263240      no
 10: 800010L4        VTL Backup 800010L4 3833926487      no
 11: 800011L4        VTL Backup 800011L4 3800372080      no
 12: 800012L4        VTL Backup 800012L4 3817149271      no
 13: 800013L4        VTL Backup 800013L4 3850703703      no
 14: 800014L4        VTL Backup 800014L4 4136089069      no
 15: 800015L4        VTL Backup 800015L4 3901035327      no
 16: 800016L4        VTL Backup 800016L4 3548713960      no
 17: 800017L4        VTL Backup 800017L4 3867480895      no
 18: 800018L4        VTL Backup 800018L4 3884258111      no
```

Figure 82: Standard *nsrjb* output

If you have multiple jukeboxes defined in your NetWorker environment and invoke *nsrjb* without specifying the jukebox to run against, *nsrjb* will become an interactive command and prompt you to specify the jukebox:

```
[root@centaur mhvtl]# nsrjb
          1:      VTL1      [enabled]
          2:      VTL2      [enabled]
No jukebox selected.
Please select a jukebox to use:? [1]
```

Figure 83: Invoking nsrjb without options in a multi-jukebox environment

To avoid this, you can invoke nsrjb as follows:

```
# nsrjb -j jbName
```

Where 'jbName' is the defined name of the jukebox in NetWorker. For instance:

```
[root@centaur mhvtl]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
14119:nsrjb: No volumes labeled.
slot  volume  pool  barcode  volume id  recyclable
  1:  -*                900001L4  -
  2:  -*                900002L4  -
  3:  -*                900003L4  -
  4:  -*                900004L4  -
  5:  -*                900005L4  -
  6:  -*                900006L4  -
  7:  -*                900007L4  -
  8:  -*                900008L4  -
  9:  -*                900009L4  -
 10:  -*                900010L4  -
      *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot  :
drive 2 (/dev/nst7) slot  :
```

Figure 84: Invoking nsrjb against a specific jukebox

The nsrjb utility is typically used for the following activities:

1. Showing the jukebox contents
2. Inventory operations
3. Reset operations
4. Media labelling
5. Media loading and unloading
6. Media import and export

It's important to note the difference between the first three activities listed above. To speed up operations against jukeboxes, NetWorker maintains a cache within the resource database of the current state of each jukebox. When an operation is performed against the jukebox, NetWorker updates the state cache. This avoids having to do a full SCSI state probe of the jukebox each time an operation is run, and considerably speeds up interaction with the device. Therefore:

1. Showing the jukebox contents – Displays the content of the jukebox per NetWorker's cache.
2. Inventorying operations – Refreshes the content of the jukebox cache from the SCSI bus.

3. Reset operations – Instructs the jukebox to perform a comprehensive reset in advance of a new inventory operation.

In particular, option (3) becomes necessary in situations where jukeboxes have become wedged through a hardware fault or non-NetWorker operations being performed on them, and option (2) is recommended after media is added to or removed from the jukebox.

14.1.1 Showing the jukebox contents

Invoked without any arguments, nsrjb shows the NetWorker cache of the jukebox state. Additional information can be displayed by invoking nsrjb with a '-v' (verbose) option. For instance, based on the jukeboxes cited above:

```
# nsrjb -v -j VTL1
```

```
[root@centaur mhvtl]# nsrjb -v -j VTL1
setting verbosity level to '1'

Jukebox VTL1: (Ready to accept commands)
slot  volume          used  pool      barcode  volume id  recyclable
1: 800001L4          0%  VTL Backup 800001L4 3515159528 no
2: 800002L4          0%  VTL Backup 800002L4 3682931616 no
3: 800003L4          0%  VTL Backup 800003L4 3699708832 no
4: 800004L4          0%  VTL Backup 800004L4 3666154400 no
5: 800005L4          0%  VTL Backup 800005L4 3716486024 no
6: 800006L4          0%  VTL Backup 800006L4 3750040456 no
7: 800007L4          0%  VTL Backup 800007L4 3766817648 no
8: 800008L4          0%  VTL Backup 800008L4 3783594864 no
9: 800009L4          0%  VTL Backup 800009L4 3733263240 no
10: 800010L4          0%  VTL Backup 800010L4 3833926487 no
11: 800011L4          0%  VTL Backup 800011L4 3800372080 no
12: 800012L4          0%  VTL Backup 800012L4 3817149271 no
13: 800013L4          0%  VTL Backup 800013L4 3850703703 no
14: 800014L4          0%  VTL Backup 800014L4 4136089069 no
15: 800015L4          0%  VTL Backup 800015L4 3901035327 no
16: 800016L4          0%  VTL Backup 800016L4 3548713960 no
17: 800017L4          0%  VTL Backup 800017L4 3867480895 no
18: 800018L4          0%  VTL Backup 800018L4 3884258111 no
19: 800019L4          0%  VTL Backup 800019L4 3481605120 no
```

Figure 85: Using nsrjb with the verbose flag

Compared to the output shown in Figure 82, this provides the volume %used field for each volume in the jukebox, making the output considerably more meaningful.

14.1.2 Jukebox Inventory

The jukebox inventory can be refreshed by using the '-I' option. If you're using a jukebox with a barcode reader, you should eschew a plain '-I' option in favour of the alternate fast inventory operation, '-II'.

The '-II' option instructs NetWorker to only inventory those volumes that have a recognised barcode in the media database. This avoids a scenario where NetWorker loads unlabelled volumes into tape drives and goes through a tedious attempted label-read operation on each tape. It's the option you would typically use when you've imported previously written media into the jukebox.

```
# nsrjb -II -j jbName
```

For instance:


```
[root@centaur ~]# nsrjb -II -j VTL1
01/02/15 20:39:58.304184 Info: Updated volume `800001L4' in slot 1.
01/02/15 20:39:58.304425 Info: Updated volume `800002L4' in slot 2.
01/02/15 20:39:58.304434 Info: Updated volume `800003L4' in slot 3.
01/02/15 20:39:58.304441 Info: Updated volume `800004L4' in slot 4.
01/02/15 20:39:58.304447 Info: Updated volume `800005L4' in slot 5.
01/02/15 20:39:58.304453 Info: Updated volume `800006L4' in slot 6.
01/02/15 20:39:58.304459 Info: Updated volume `800007L4' in slot 7.
01/02/15 20:39:58.304465 Info: Updated volume `800008L4' in slot 8.
01/02/15 20:39:58.304472 Info: Updated volume `800009L4' in slot 9.
01/02/15 20:39:58.304478 Info: Updated volume `800010L4' in slot 10.
01/02/15 20:39:58.304484 Info: Updated volume `800011L4' in slot 11.
01/02/15 20:39:58.304491 Info: Updated volume `800012L4' in slot 12.
01/02/15 20:39:58.304497 Info: Updated volume `800013L4' in slot 13.
01/02/15 20:39:58.304503 Info: Updated volume `800014L4' in slot 14.
01/02/15 20:39:58.304509 Info: Updated volume `800015L4' in slot 15.
01/02/15 20:39:58.304515 Info: Updated volume `800016L4' in slot 16.
01/02/15 20:39:58.304521 Info: Updated volume `800017L4' in slot 17.
01/02/15 20:39:58.304527 Info: Updated volume `800018L4' in slot 18.
01/02/15 20:39:58.304533 Info: Updated volume `800019L4' in slot 19.
01/02/15 20:39:58.304539 Info: Updated volume `800020L4' in slot 20.
```

Figure 86: Running a fast inventory operation

The above output demonstrates NetWorker detecting volumes that already have matching barcodes in the media database. Run against a jukebox with volumes that haven't been labelled gives a completely different result:

```
[root@centaur ~]# nsrjb -II -j VTL2
01/02/15 20:41:46.278797 Info: Fast Inventory cannot identify volume in slot `1'.
01/02/15 20:41:46.279286 Info: Fast Inventory cannot identify volume in slot `2'.
01/02/15 20:41:46.279297 Info: Fast Inventory cannot identify volume in slot `3'.
01/02/15 20:41:46.279305 Info: Fast Inventory cannot identify volume in slot `4'.
01/02/15 20:41:46.279313 Info: Fast Inventory cannot identify volume in slot `5'.
01/02/15 20:41:46.279321 Info: Fast Inventory cannot identify volume in slot `6'.
01/02/15 20:41:46.279329 Info: Fast Inventory cannot identify volume in slot `7'.
01/02/15 20:41:46.279337 Info: Fast Inventory cannot identify volume in slot `8'.
01/02/15 20:41:46.279346 Info: Fast Inventory cannot identify volume in slot `9'.
01/02/15 20:41:46.279354 Info: Fast Inventory cannot identify volume in slot `10'.
01/02/15 20:41:46.279566 Error: Fast Inventory fail to identify some or all of the selected slots.
Jukebox operation finished with status: succeeded
69411:nsrjb: Jukebox command terminated with errors but completed successfully.
```

Figure 87: Using fast inventory when volumes have not previously been labelled

This leads us to the first key recommendation with nsrjb – any time you perform an activity beyond the basic display of jukebox contents, you should use '-vvv' in the command line. This provides a high level of verbosity. For instance:

```
# nsrjb -II -j VTL2 -vvv
```

```
[root@centaur ~]# nsrjb -II -vvv -j VTL2
setting verbosity level to '3'
01/02/15 20:47:33.110322 Info: The media data base has no volume with barcode = '900001L4'.
01/02/15 20:47:33.110541 Info: Fast Inventory cannot identify volume in slot '1'.
01/02/15 20:47:33.110552 Info: The media data base has no volume with barcode = '900002L4'.
01/02/15 20:47:33.110561 Info: Fast Inventory cannot identify volume in slot '2'.
01/02/15 20:47:33.110570 Info: The media data base has no volume with barcode = '900003L4'.
01/02/15 20:47:33.110579 Info: Fast Inventory cannot identify volume in slot '3'.
01/02/15 20:47:33.110614 Info: The media data base has no volume with barcode = '900004L4'.
01/02/15 20:47:33.110651 Info: Fast Inventory cannot identify volume in slot '4'.
01/02/15 20:47:33.110659 Info: The media data base has no volume with barcode = '900005L4'.
01/02/15 20:47:33.110668 Info: Fast Inventory cannot identify volume in slot '5'.
01/02/15 20:47:33.110676 Info: The media data base has no volume with barcode = '900006L4'.
01/02/15 20:47:33.110685 Info: Fast Inventory cannot identify volume in slot '6'.
01/02/15 20:47:33.110693 Info: The media data base has no volume with barcode = '900007L4'.
01/02/15 20:47:33.110702 Info: Fast Inventory cannot identify volume in slot '7'.
01/02/15 20:47:33.110710 Info: The media data base has no volume with barcode = '900008L4'.
01/02/15 20:47:33.110721 Info: Fast Inventory cannot identify volume in slot '8'.
01/02/15 20:47:33.110729 Info: The media data base has no volume with barcode = '900009L4'.
01/02/15 20:47:33.110738 Info: Fast Inventory cannot identify volume in slot '9'.
01/02/15 20:47:33.110746 Info: The media data base has no volume with barcode = '900010L4'.
01/02/15 20:47:33.110754 Info: Fast Inventory cannot identify volume in slot '10'.
01/02/15 20:47:33.110873 Error: Fast Inventory fail to identify some or all of the selected slots.
Jukebox operation finished with status: succeeded
69411:nsrjb: Jukebox command terminated with errors but completed successfully.
```

Figure 88: Using nsrjb with higher levels of verbosity

The importance of '-vvv' becomes apparent if we switch from fast inventory to regular inventory mode:

```
# nsrjb -vvv -I -j VTL2
```

For instance:

```
[root@centaur ~]# nsrjb -vvv -I -j VTL2
setting verbosity level to '3'
01/02/15 20:49:29.914884 Info: The media data base has no volume with barcode = '900001L4'.
01/02/15 20:49:29.914964 Info: The media data base has no volume with barcode = '900002L4'.
01/02/15 20:49:29.914977 Info: The media data base has no volume with barcode = '900003L4'.
01/02/15 20:49:29.914983 Info: The media data base has no volume with barcode = '900004L4'.
01/02/15 20:49:29.914990 Info: The media data base has no volume with barcode = '900005L4'.
01/02/15 20:49:29.914996 Info: The media data base has no volume with barcode = '900006L4'.
01/02/15 20:49:29.915002 Info: The media data base has no volume with barcode = '900007L4'.
01/02/15 20:49:29.915008 Info: The media data base has no volume with barcode = '900008L4'.
01/02/15 20:49:29.915015 Info: The media data base has no volume with barcode = '900009L4'.
01/02/15 20:49:29.915021 Info: The media data base has no volume with barcode = '900010L4'.
01/02/15 20:49:29.915045 Info: Preparing to load volume '-' from slot 1 into device '/dev/nst6'.
01/02/15 20:49:29.915054 Info: Loading volume '-' from slot '1' into device '/dev/nst6'.
01/02/15 20:49:29.915061 Info: Load sleep for 5 seconds.
01/02/15 20:49:29.915069 Info: Performing operation 'Verify label' on device '/dev/nst6'.
01/02/15 20:49:29.915079 Info: Operation 'Verify label' in progress on device '/dev/nst6'
01/02/15 20:49:29.915112 Info: Cannot read the current volume label: no tape label found
01/02/15 20:49:29.915119 Info: Assuming the volume is unlabeled.
01/02/15 20:49:29.915126 Info: Performing operation 'Eject' on device '/dev/nst6'.
01/02/15 20:49:29.915133 Info: Operation 'Eject' in progress on device '/dev/nst6'
01/02/15 20:49:29.915141 Info: Eject sleep for 5 seconds.
01/02/15 20:49:29.915149 Info: Preparing to unload volume '-' from device '/dev/nst6' to slot 1.
01/02/15 20:49:29.915154 Info: Unloading volume '-' from device '/dev/nst6' to slot 1.
01/02/15 20:49:29.915161 Info: Unload sleep for 5 seconds.
```

Figure 89: Performing a slow inventory using extended verbose mode

You'll note in this mode that nsrjb becomes considerably chattier – the advantage being it reveals exactly what operations it's going to perform. It outlines, for instance, each load, verify label and eject operation as it is performed.

14.1.2.1 Granular Control of the Jukebox

Whenever you're performing operations with `nsrjb`, you can narrow down tape/volume operations by:

- Slot
- Device
- Volume

These options are specified as:

Table 4: Options for limiting the extent of an `nsrjb` operation to particular volumes, slots or devices

Option	Description
<code>-S range</code>	Specify a slot range. May be a single slot or a dashed range (e.g., 3I-45). You can specify multiple slot options (e.g., <code>-S 7 -S 9-16</code>).
<code>-f device</code>	Perform the requested operation using a specifically nominated device. This can be used for loading and unloading media as well as labelling media.
<code>volumeName</code>	You can also target specific volumes for <code>nsrjb</code> operations. The volume name(s) must be listed at the very end of the command.

For example, consider a jukebox where we want to inventory (with an actual load and label read operation taking place) just the tapes in slots 1, 3, 4 and 5. The following command could be used:

```
# nsrjb -Ivvv -j VTL2 -S 1 -S 3-5
```

Output from this command might resemble the following:

```
[root@centaur ~]# nsrjb -Ivvv -j VTL2 -S 1 -S 3-5
setting verbosity level to '3'
01/03/15 08:14:07.589228 Info: The media data base has no volume with barcode = '900001L4'.
01/03/15 08:14:07.589355 Info: The media data base has no volume with barcode = '900003L4'.
01/03/15 08:14:07.589363 Info: The media data base has no volume with barcode = '900004L4'.
01/03/15 08:14:07.589370 Info: The media data base has no volume with barcode = '900005L4'.
01/03/15 08:14:07.589379 Info: Preparing to load volume '-' from slot 1 into device '/dev/nst7'.
01/03/15 08:14:07.589388 Info: Loading volume '-' from slot '1' into device '/dev/nst7'.
01/03/15 08:14:07.589395 Info: Load sleep for 5 seconds.
01/03/15 08:14:07.589402 Info: Performing operation 'Verify label' on device '/dev/nst7'.
01/03/15 08:14:17.590422 Info: Operation 'Verify label' in progress on device '/dev/nst7'
01/03/15 08:14:17.590464 Info: Cannot read the current volume label: no tape label found
01/03/15 08:14:17.590471 Info: Assuming the volume is unlabeled.
01/03/15 08:14:17.590478 Info: Performing operation 'Eject' on device '/dev/nst7'.
01/03/15 08:14:17.590485 Info: Operation 'Eject' in progress on device '/dev/nst7'
01/03/15 08:14:17.590492 Info: Eject sleep for 5 seconds.
01/03/15 08:14:17.590500 Info: Preparing to unload volume '-' from device '/dev/nst7' to slot 1.
01/03/15 08:14:27.591198 Info: Unloading volume '-' from device '/dev/nst7' to slot 1.
01/03/15 08:14:27.591229 Info: Unload sleep for 5 seconds.
01/03/15 08:14:27.591238 Info: Preparing to load volume '-' from slot 3 into device '/dev/nst7'.
```

Figure 90: Limiting a jukebox inventory to specific slots

Equally, if we wanted to repeat the same operation using only a specific device (e.g., `/dev/nst6`), we could issue the command:

```
# nsrjb -Ivvv -j VTL2 -S 1 -S 3-5 -f /dev/nst6
```

This would execute similarly to the example below:

```
[root@centaur ~]# nsrjb -Ivvv -j VTL2 -S 1 -S 3-5 -f /dev/nst6
setting verbosity level to '3'
01/03/15 08:25:54.371862 Info: The media data base has no volume with barcode = '900001L4'.
01/03/15 08:25:54.372010 Info: The media data base has no volume with barcode = '900003L4'.
01/03/15 08:25:54.372018 Info: The media data base has no volume with barcode = '900004L4'.
01/03/15 08:25:54.372025 Info: The media data base has no volume with barcode = '900005L4'.
01/03/15 08:25:54.372033 Info: Preparing to load volume '-' from slot 1 into device '/dev/nst6'.
01/03/15 08:25:54.372042 Info: Loading volume '-' from slot '1' into device '/dev/nst6'.
01/03/15 08:25:54.372048 Info: Load sleep for 5 seconds.
01/03/15 08:25:54.372055 Info: Performing operation 'Verify label' on device '/dev/nst6'.
01/03/15 08:26:04.372867 Info: Operation 'Verify label' in progress on device '/dev/nst6'
01/03/15 08:26:04.372898 Info: Cannot read the current volume label: no tape label found
01/03/15 08:26:04.372905 Info: Assuming the volume is unlabeled.
01/03/15 08:26:04.372912 Info: Performing operation 'Eject' on device '/dev/nst6'.
01/03/15 08:26:04.372919 Info: Operation 'Eject' in progress on device '/dev/nst6'
01/03/15 08:26:04.372926 Info: Eject sleep for 5 seconds.
01/03/15 08:26:04.372934 Info: Preparing to unload volume '-' from device '/dev/nst6' to slot 1.
01/03/15 08:26:14.374580 Info: Unloading volume '-' from device '/dev/nst6' to slot 1.
```

Figure 91: Limiting jukebox operations to a particular device

We'll cover an example of referencing volume names directly in section 14.1.4 (Labelling and Relabeling Media).

14.1.3 Resetting a Jukebox

A jukebox can be reset using one of the following two options:

```
# nsrjb -HE [-j jbName] [-vvv]
```

or

```
# nsrjb -HHE [-vvv] [-j jbName]
```

The first reset command attempts to unload all devices in the jukebox *and* throws away the NetWorker cache information for the jukebox, forcing a refresh. The second reset command attempts a *forced* unload of all the devices in the jukebox and throws away the NetWorker cache for the jukebox.

The -E option above instructs the jukebox to throw away any slot cache information and refresh it. While theoretically the -H and -E options can be used independently, there is little practical value in doing so.

The primary difference between -H -HH is how low-level a reset command is issued against the tape drive(s) in the tape library. A -H option is less interruptive to the tape drives; a -HH option is intended to try to forcibly reset a tape drive even if it's busy.

If there are no tapes in any drives, a reset operation is reasonably straight-forward:

```
# nsrjb -HEvvv -j VTL2
```

```
[root@centaur ~]# nsrjb -HEvvv -j VTL2
setting verbosity level to '3'
01/02/15 21:02:56.612009 Info: Preparing to perform reset operation on jukebox 'VTL2'.
01/02/15 21:02:56.612164 Info: Operation 'initialize elements status' is in progress on jukebox 'VTL2'.
01/02/15 21:02:56.612175 Info: There is nothing to unload in jukebox 'VTL2'.
Jukebox operation finished with status: succeeded
```

Figure 92: Basic jukebox reset command

However, if there's a tape in any drive, the output from a reset command is as follows:

```
[root@centaur ~]# nsrjb -HHEvvv -j VTL2
setting verbosity level to '3'
01/02/15 21:05:32.244722 Info: Preparing to perform reset operation on jukebox 'VTL2'.
01/02/15 21:05:32.244827 Info: Operation 'initialize elements status' is in progress on jukebox 'VTL2'.
01/02/15 21:05:42.245667 Info: Performing operation 'Eject' on device '/dev/nst6'.
01/02/15 21:05:42.245717 Info: Operation 'Eject' in progress on device '/dev/nst6'.
01/02/15 21:05:42.245727 Info: Eject sleep for 5 seconds.
01/02/15 21:05:42.245736 Info: Preparing to unload volume '-' from device '/dev/nst6' to slot 1.
01/02/15 21:05:42.245744 Info: Unloading volume '-' from device '/dev/nst6' to slot 1.
01/02/15 21:05:42.245751 Info: Unload sleep for 5 seconds.
Jukebox operation finished with status: succeeded
```

Figure 93: Jukebox reset command when there are volumes to unload

One thing you should be mindful of is how long a reset operation might take on any autochangers you have configured. For virtual tape libraries this of course will be relatively short, but for physical tape libraries that time may vary from between a minute to ten minutes depending on the brand, size and number of tape drives¹⁵.

14.1.4 Labelling and Relabeling Media

There are two command line options in particular for nsrjb that are used in the labelling and relabeling of media. These are:

- -L – Label a volume
- -R – Recycle a volume

A typical invocation of nsrjb to label a volume is as follows:

```
# nsrjb -j jbName -L -b poolName -S slotRange
```

For instance, the media in VTL2 is currently unlabelled based on the lab examples we've done so far. If I wanted to label the first three volumes in the library into the *VTL Backup* pool, I could use the following command:

```
# nsrjb -j VTL2 -Lv vv -b "VTL Backup" -S 1-3
```

Note I've included the '-vvv' option again. It makes monitoring a label or relabel operation so much simpler.

This would produce output such as the following:

```
[root@centaur ~]# nsrjb -j VTL2 -Lv vv -b "VTL Backup" -S 1-3
setting verbosity level to '3'
01/03/15 08:36:41.720947 Info: Preparing to load volume '-' from slot 1 into device '/dev/nst7'.
01/03/15 08:36:41.721278 Info: Loading volume '-' from slot '1' into device '/dev/nst7'.
01/03/15 08:36:41.721287 Info: Load sleep for 5 seconds.
01/03/15 08:36:41.721294 Info: Performing operation 'Verify label' on device '/dev/nst7'.
01/03/15 08:36:51.721975 Info: Operation 'Verify label' in progress on device '/dev/nst7'.
01/03/15 08:36:51.722022 Info: Cannot read the current volume label: no tape label found
01/03/15 08:36:51.722031 Info: nsrmmgd assumes the volume is unlabeled and will write a new label.
01/03/15 08:36:51.722040 Info: Performing operation 'Label without mount' on device '/dev/nst7'.
01/03/15 08:36:51.722048 Info: Operation 'Label without mount' in progress on device '/dev/nst7'.
01/03/15 08:36:51.722056 Info: Label: '900001L4', pool: 'VTL Backup', capacity: '<NULL>'.
01/03/15 08:36:51.722064 Info: Performing operation 'Eject' on device '/dev/nst7'.
01/03/15 08:36:51.722071 Info: Operation 'Eject' in progress on device '/dev/nst7'.
01/03/15 08:36:51.722182 Info: Eject sleep for 5 seconds.
01/03/15 08:36:51.722201 Info: Preparing to unload volume '900001L4' from device '/dev/nst7' to slot 1.
01/03/15 08:37:01.722944 Info: Unloading volume '900001L4' from device '/dev/nst7' to slot 1.
01/03/15 08:37:01.722967 Info: Unload sleep for 5 seconds.
```

Figure 94: Performing a media label operation

¹⁵ The best way understand this of course is to actually physically *watch* the autochanger while a reset operation is being performed.

After the operation completes, we can run `nsrjb` for the nominated jukebox and verify the volumes are now labelled and available:

```
# nsrjb -j VTL2
```

```
[root@centaur ~]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
slot volume          pool          barcode    volume id    recyclable
  1: 900001L4        VTL Backup    900001L4    4238806889    no
  2: 900002L4        VTL Backup    900002L4    4222029695    no
  3: 900003L4        VTL Backup    900003L4    4205252501    no
  4: -*              900004L4     -
  5: -*              900005L4     -
  6: -*              900006L4     -
  7: -*              900007L4     -
  8: -*              900008L4     -
  9: -*              900009L4     -
 10: -*              900010L4     -

      *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot : service mode
drive 2 (/dev/nst7) slot :
[root@centaur ~]#
```

Figure 95: Viewing the status of the jukebox after a label operation

The other labelling command is the `-R` option for volume recycling. Normally you shouldn't need to use this option – NetWorker handles volume recycling on an as-needed basis if you've configured it correctly¹⁶.

Consider the VTL2 jukebox used in these examples again. If we realised the volume in slot 3 should have actually been labelled into the "VTL Clone" pool, we can relabel it into that pool as follows:

```
# nsrjb -j VTL2 -LRvvv -b "VTL Clone" -S 3
```

Note that NetWorker will prompt to confirm if you really want to do this:

```
[root@centaur ~]# nsrjb -j VTL2 -LRvvv -b "VTL Clone" -S 3
setting verbosity level to '3'
Label '900003L4' is a valid NetWorker label. Overwrite it with a new label? y
01/03/15 08:44:02.239190 Info: Preparing to load volume '900003L4' from slot 3 into device '/dev/nst7'.
01/03/15 08:44:02.239346 Info: Loading volume '900003L4' from slot '3' into device '/dev/nst7'.
01/03/15 08:44:02.239354 Info: Load sleep for 5 seconds.
01/03/15 08:44:02.239361 Info: Performing operation 'Verify label' on device '/dev/nst7'.
01/03/15 08:44:12.240083 Info: Operation 'Verify label' in progress on device '/dev/nst7'
01/03/15 08:44:12.240108 Info: Performing operation 'Label without mount' on device '/dev/nst7'.
01/03/15 08:44:12.240116 Info: Operation 'Label without mount' in progress on device '/dev/nst7'
01/03/15 08:44:12.240124 Info: Recycling volume '900003L4' to pool 'VTL Clone'
01/03/15 08:44:12.240131 Info: Performing operation 'Eject' on device '/dev/nst7'.
01/03/15 08:44:12.240138 Info: Operation 'Eject' in progress on device '/dev/nst7'
01/03/15 08:44:12.240145 Info: Eject sleep for 5 seconds.
01/03/15 08:44:22.241454 Info: Preparing to unload volume '900003L4' from device '/dev/nst7' to slot 3.
01/03/15 08:44:22.241811 Info: Unloading volume '900003L4' from device '/dev/nst7' to slot 3.
01/03/15 08:44:22.241821 Info: Unload sleep for 5 seconds.
Jukebox operation finished with status: succeeded
```

Figure 96: Recycling a volume into another pool

¹⁶ Note that I am not referring to auto media management (AMM) for tape libraries. The use of AMM is unrelated to recycling operations.

Recycling operations don't actually require you to specify a pool name. If you don't, the volume will be recycled back into the pool it was currently labelled in.

You may wonder what the difference between the `-R` and `-L` options are. They become more evident if we actually perform the operation against a volume that's flagged as recyclable. For instance, the volume in slot 42 in VTL1 on this lab server is recyclable, and so issuing the command:

```
# nsrjb -j VTL1 -LRvvv -S 42
```

Results in an immediate recycle, without any prompting.

```
[root@centaur ~]# nsrjb -j VTL1 | grep '(R)'
42: 800042L4(R)    VTL Backup  800042L4  4203025071      yes
[root@centaur ~]# nsrjb -j VTL1 -LRvvv -S 42
setting verbosity level to '3'
01/03/15 08:48:48.583426 Info: Preparing to load volume '800042L4' from slot 42 into device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
01/03/15 08:48:48.583574 Info: Loading volume '800042L4' from slot '42' into device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
01/03/15 08:48:48.583607 Info: Load sleep for 5 seconds.
01/03/15 08:48:48.583616 Info: Performing operation 'Verify label' on device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
01/03/15 08:48:58.584412 Info: Operation 'Verify label' in progress on device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
01/03/15 08:48:58.584446 Info: Performing operation 'Label without mount' on device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
01/03/15 08:48:58.584455 Info: Operation 'Label without mount' in progress on device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
01/03/15 08:48:58.584462 Info: Recycling volume '800042L4'
01/03/15 08:48:58.584469 Info: Performing operation 'Eject' on device '/dev/tape/by-id/scsi-350223344ab000400-nst'.
```

Figure 97: Using the recycle option against a recyclable volume

14.1.5 Loading and Unloading Volumes

For the most part, NetWorker automatically handles the loading and unloading of volumes within jukeboxes. Indeed, the standard settings in NetWorker these days will have it automatically unload volumes after they've been idle for 10 minutes.

There are some times though when you may want to manually load or unload a volume:

- Loading:
 - When you're preparing to conduct a test and you want the media online immediately
 - When you're performing an operation and you want to override NetWorker's default volume selection criteria (e.g., next volume to use for a backup) by loading a specific volume
 - When you want to do a bootstrap recovery using a tape library¹⁷
- Unloading:
 - When you're preparing to export media from the tape library
 - When you want to change the mode on volumes¹⁸

The command line options for load and unload are:

- `-l` – Load a volume
- `-u` – Unload a volume
- `-nl` – Load but do not attempt to mount a volume

¹⁷ This is a special operation compared to normal load operations.

¹⁸ As we saw in the *nsrmm* section, you can't perform a volume mode change (e.g., setting it to 'readonly') while it is mounted in a drive.

Examples of these operations are as follows – the ‘-vvv’ option has been left off for demonstration purposes:

```
# nsrjb -l -j VTL2 -S 1

[root@centaur ~]# nsrjb -l -j VTL2 -S 1
01/03/15 08:58:42.708385 Info: Operation 'Mount' in progress on device '/dev/nst7'
Jukebox operation finished with status: succeeded
[root@centaur ~]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
slot  volume          pool          barcode    volume id      recyclable
  1:  900001L4        VTL Backup   900001L4    4238806889     no
  2:  900002L4        VTL Backup   900002L4    4222029695     no
  3:  900003L4        VTL Clone    900003L4    4188475684     no
  4:  -*              900004L4     -
  5:  -*              900005L4     -
  6:  -*              900006L4     -
  7:  -*              900007L4     -
  8:  -*              900008L4     -
  9:  -*              900009L4     -
 10:  -*              900010L4     -
      *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot  :
drive 2 (/dev/nst7) slot  1: 900001L4 (mounted)
[root@centaur ~]#
```

Figure 98: Performing a volume load operation

```
# nsrjb -u -j VTL2 900001L4

[root@centaur ~]# nsrjb -u -j VTL2 900001L4
01/03/15 09:01:14.334998 Info: Operation 'Eject' in progress on device '/dev/nst7'
Jukebox operation finished with status: succeeded
[root@centaur ~]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
slot  volume          pool          barcode    volume id      recyclable
  1:  900001L4        VTL Backup   900001L4    4238806889     no
  2:  900002L4        VTL Backup   900002L4    4222029695     no
  3:  900003L4        VTL Clone    900003L4    4188475684     no
  4:  -*              900004L4     -
  5:  -*              900005L4     -
  6:  -*              900006L4     -
  7:  -*              900007L4     -
  8:  -*              900008L4     -
  9:  -*              900009L4     -
 10:  -*              900010L4     -
      *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot  :
drive 2 (/dev/nst7) slot  :
[root@centaur ~]#
```

Figure 99: Performing a volume unload operation, by volume name

You’ll notice in the above command we unloaded the volume by *name* rather than by slot number. (Note there is no command line switch before a volume name.) If you wanted to unload a volume

by *slot number*, you'd have to be careful to use the same slot number that is shown against the device in the nsrjb output.

```
# nsrjb -nl -j VTL2 -S 1 -f /dev/nst7

[root@centaur ~]# nsrjb -nl -j VTL2 -S 1 -f /dev/nst7
Jukebox operation finished with status: succeeded
[root@centaur ~]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
slot volume pool barcode volume id recyclable
1: 900001L4 VTL Backup 900001L4 4238806889 no
2: 900002L4 VTL Backup 900002L4 4222029695 no
3: 900003L4 VTL Clone 900003L4 4188475684 no
4: -* 900004L4 -
5: -* 900005L4 -
6: -* 900006L4 -
7: -* 900007L4 -
8: -* 900008L4 -
9: -* 900009L4 -
10: -* 900010L4 -
    *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot :
drive 2 (/dev/nst7) slot 1: 900001L4
```

Figure 100: Loading a volume without mounting it

Notice the difference in the output between this nsrjb command and the one used in Figure 98 – in that nsrjb output, we see:

```
drive 2 (/dev/nst7) slot 1: 900001L4 (mounted)
```

However, in the most recent nsrjb command output, we only saw:

```
drive 2 (/dev/nst7) slot 1: 900001L4
```

That's NetWorker's way of telling you the volume had been loaded, but it hasn't attempted to read the volume label.

The load-without-mount operation is most commonly used in the following circumstances:

- When preparing to do a bootstrap recovery after a loss of (*at least* the media database)
- When preparing to scan a volume in from another NetWorker server

In both of those scenarios, NetWorker won't *recognise* the volume label it reads from the tape. By default, when this happens, NetWorker ejects the tape – something that is undesirable if you actually want to read from it or scan it.

14.1.6 Exporting and Importing Media

Most tape libraries (even virtual ones) have the concept of a Cartridge Access Port (CAP) or mail slot. If you're using a physical tape library the chances are high you'll need to periodically remove tapes from the library to free up capacity (or to off-site them), and that you'll need to add media to the library¹⁹.

¹⁹ Opening the library door and manually adding tapes to slots or removing them results in extremely chaotic operations unless you do a full reset and inventory afterwards. It's not recommended.

Even if you're using a VTL, you may want to make use of the virtual CAP/Mail Slot for the following reasons:

- The VTL may support a vaulting function, where more media is defined than is loaded in the library at any given time
- Volumes in the CAP or Mail Slot are not eligible for NetWorker to use for operations until they've been imported into the main slot range, which neatly allows you to get particular volumes temporarily out of the way.

The CLI options for these functions are:

- -w – Withdraw a tape from the library and place it in the CAP
- -d – Deposit a tape from the CAP into the library

For instance:

```
# nsrjb -w -j VTL2 -S 1
```

Would withdraw the tape in slot 1 of VTL2 into VTL2's CAP.

```
[root@centaur ~]# nsrjb -w -j VTL2 -S 1
01/03/15 09:20:02.352052 Info: Operation 'read elements status' is in progress on jukebox 'VTL2'.
01/03/15 09:20:02.352167 Info: Moving media from slot 1 to port 1.
01/03/15 09:20:02.352175 Info: Withdrawing 1 cartridges.
01/03/15 09:20:02.352181 Info: Remove any cartridges from the ports.
Jukebox operation finished with status: succeeded
```

Figure 101: Withdrawing a volume into the CAP

After the volume has been withdrawn, the output from 'nsrjb' shows a completely empty slot 1:

```
[root@centaur ~]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
slot  volume          pool          barcode    volume id    recyclable
 1:
 2: 900002L4          VTL Backup    900002L4    4222029695    no
 3: 900003L4          VTL Clone     900003L4    4188475684    no
 4: -*
 5: -*
 6: -*
 7: -*
 8: -*
 9: -*
10: -*
    *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot :
drive 2 (/dev/nst7) slot :
```

Figure 102: Jukebox state after a withdraw operation

The deposit function works is (mostly) designed for interactive prompting for scenarios where an operator may be loading more tapes into the library than there are slots in the CAP. For instance:

```
# nsrjb -d -j VTL2 -S 1
```

```
[root@centaur ~]# nsrjb -d -j VTL2 -S 1
Load the cartridges into the ports, and enter Yes to continue. yes
01/03/15 09:26:36.529133 Info: Operation 'read elements status' is in progress on jukebox 'VTL2'.
01/03/15 09:26:36.529344 Info: Moving media from port 1 to slot 1.
01/03/15 09:26:36.529353 Info: Depositing 1 cartridges.
Jukebox operation finished with status: succeeded
[root@centaur ~]# nsrjb -j VTL2

Jukebox VTL2: (Ready to accept commands)
slot volume pool barcode volume id recyclable
1: 900001L4 VTL Backup 900001L4 4238806889 no
2: 900002L4 VTL Backup 900002L4 4222029695 no
3: 900003L4 VTL Clone 900003L4 4188475684 no
4: -* 900004L4 -
5: -* 900005L4 -
6: -* 900006L4 -
7: -* 900007L4 -
8: -* 900008L4 -
9: -* 900009L4 -
10: -* 900010L4 -
      *not registered in the NetWorker media data base

drive 1 (/dev/nst6) slot :
drive 2 (/dev/nst7) slot :
```

Figure 103: Performing a deposit operation

You'll notice there was no inventory operation performed after that deposit, yet NetWorker recognised the volume. When the "match barcode labels" option is turned on in NetWorker, newer versions of NetWorker will automatically inventory volumes based on matching barcodes when they are deposited into the library.

14.2 Low level interaction

So far our activities with NetWorker and jukeboxes have been conducted with *nsrjb*, which is the preferred way to work with it. However, there are times you want to dig a little lower and work with or interrogate a library directly.

CAUTION - Lab Exercises

All the exercises described in this low-level interaction section should be performed first in a lab if you are not familiar with them. Exercises may reset or hang SCSI busses if performed incorrectly, or against an active bus. Additionally, exercises may result in media ending up in locations not known about by NetWorker. Proceed with this topic with **caution**.

Note that all of the commands in this section can be used regardless of whether NetWorker is up or down.

To be able to do this, we first have to know the SCSI target/LUN/bus numbers of our tape libraries. Thankfully, NetWorker has a utility to provide this information to us, and that's *inquire*.

The two most common ways of invoking *inquire* are:

```
# inquire -l
# inquire -lp
```

The first *inquire* command lists all SCSI targets. The second lists all SCSI targets and outputs (where possible and supported) using persistent device names. To understand the difference, consider the output from both:

```
# inquire -l
```

```
[root@centaur ~]# inquire -l

-l flag found: searching all LUNs, which may take over 10 minutes per adapter
for some fibre channel adapters. Please be patient.

(using name lookup)
77662:inquire: CDI warning: Using old-style passthrough via st driver
77662:inquire: CDI warning: Using old-style passthrough via st driver
scsidev@0.0.0:NECVMWareVMware IDE CDR101.00|CD-ROM, /dev/sg0
scsidev@1.0.0:VMware Virtual disk 1.0 |Disk, /dev/sg1
scsidev@1.1.0:VMware Virtual disk 1.0 |Disk, /dev/sg2
scsidev@1.2.0:VMware Virtual disk 1.0 |Disk, /dev/sg3
scsidev@2.0.0:STK L700 0105|Autochanger (Jukebox), /dev/sg12
S/N: XYZZY_A
scsidev@2.1.0:IBM ULT3580-TD4 0105|Tape, /dev/nst0
S/N: XYZZY_A1
ATNN=IBM ULT3580-TD4 XYZZY_A1
WWNN=50223344AB000100
Can Encrypt & Decrypt
scsidev@2.2.0:IBM ULT3580-TD4 0105|Tape, /dev/nst1
S/N: XYZZY_A2
ATNN=IBM ULT3580-TD4 XYZZY_A2
WWNN=50223344AB000200
Can Encrypt & Decrypt
```

Figure 104: Standard inquire output

On the other hand:

```
# inquire -lp

[root@centaur ~]# inquire -lp

-l flag found: searching all LUNs, which may take over 10 minutes per adapter
for some fibre channel adapters. Please be patient.

(using name lookup)
scsidev@2.0.0:STK L700 0105|Autochanger (Jukebox), /dev/tape/by-id/scsi-SSTK_L700_XYZZY_A
S/N: XYZZY_A
scsidev@2.3.0:IBM ULT3580-TD4 0105|Tape, /dev/tape/by-id/scsi-350223344ab000300-nst
S/N: XYZZY_A3
ATNN=IBM ULT3580-TD4 XYZZY_A3
WWNN=50223344AB000300
Can Encrypt & Decrypt
scsidev@2.4.0:IBM ULT3580-TD4 0105|Tape, /dev/tape/by-id/scsi-350223344ab000400-nst
S/N: XYZZY_A4
ATNN=IBM ULT3580-TD4 XYZZY_A4
WWNN=50223344AB000400
Can Encrypt & Decrypt
scsidev@2.5.0:IBM ULT3580-TD4 0105|Tape, /dev/tape/by-id/scsi-350223344ab000500-nst
S/N: XYZZY_A5
ATNN=IBM ULT3580-TD4 XYZZY_A5
WWNN=50223344AB000500
Can Encrypt & Decrypt
```

Figure 105: The inquire command with persistent device names

In both cases, the SCSI Bus/Target/LUN information is presented in the `scsidev@b.t.l` component of the output. For instance, this tells us we have a jukebox accessible on `scsidev@2.0.0`.

14.2.1 `sjisn`

The simplest low level library operation for us to execute is `sjisn`. This reports the serial numbers for devices in the tape library *in the device order defined in the tape library*. Depending on your cabling for tape libraries (particularly physical ones), the device order within the library may not

match the actual device order presented to the operating system, and `sjisn` can help you cross-reference those hardware devices to the OS device paths. This can be particularly helpful when you're wanting to configure a jukebox via `jbconfig` or have a jukebox with devices shared across multiple hosts.

The `sjisn` command is invoked as follows:

```
# sjisn b.t.l
```

Where `b.t.l` is the matching bus, target and LUN for the library.

For instance:

```
# sjisn 2.0.0
```

```
[root@centaur ~]# sjisn 2.0.0
Serial Number data for 2.0.0 (STK      L700      ):
  Library:
    Serial Number: XYZZY_A
  Drive at element address 500:
    SCSI-3 Device Identifiers:
      ATNN=IBM      ULT3580-TD4      XYZZY_A1
  Drive at element address 501:
    SCSI-3 Device Identifiers:
      ATNN=IBM      ULT3580-TD4      XYZZY_A2
  Drive at element address 502:
    SCSI-3 Device Identifiers:
      ATNN=IBM      ULT3580-TD4      XYZZY_A3
  Drive at element address 503:
    SCSI-3 Device Identifiers:
      ATNN=IBM      ULT3580-TD4      XYZZY_A4
  Drive at element address 504:
    SCSI-3 Device Identifiers:
      ATNN=IBM      ULT3580-TD4      XYZZY_A5
  Drive at element address 505:
    SCSI-3 Device Identifiers:
      ATNN=IBM      ULT3580-TD4      XYZZY_A6
```

Figure 106: Output from `sjisn`

From this output, we can tell for instance that the first drive defined within the library (“Drive at element address 500”) has the serial number `XYZZY_A1`, which based on the previous `inquire` output (see Figure 104) is device `/dev/nst0` on the server.

14.2.2 `sjirdtag`

For the `sjirdtag` command, we'll use the smaller VTL defined on the lab server – VTL2. The relevant `inquire` output for this is:

```
scsidev@3.0.0:STK      L700      0105|Autochanger (Jukebox), /dev/sg13
                        S/N: AYZZY_A
scsidev@3.1.0:IBM      ULT3580-TD4  0105|Tape, /dev/nst6
                        S/N: AYZZY_A1
                        ATNN=IBM      ULT3580-TD4      AYZZY_A1
                        WWNN=50223344AB000100
                        Can Encrypt & Decrypt
scsidev@3.2.0:IBM      ULT3580-TD4  0105|Tape, /dev/nst7
                        S/N: AYZZY_A2
                        ATNN=IBM      ULT3580-TD4      AYZZY_A2
                        WWNN=50223344AB000200
                        Can Encrypt & Decrypt
```

Figure 107: Inquire output showing second jukebox

This tells us the second jukebox is on `scsidev@3.0.0`.

The syntax for `sjirdtag` is:

```
# sjirdtag b.t.l
```

Where `b.t.l` is the bus, target and LUN number of the jukebox we want to probe.

For, for our lab server, the command would be:

```
# sjirdtag 3.0.0
```

```
[root@centaur ~]# sjirdtag 3.0.0
Tag Data for 3.0.0, Element Type DATA TRANSPORT:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[002]: tag_val=0 pres_val=1 med_pres=0 med_side=0
Tag Data for 3.0.0, Element Type STORAGE:
  Elem[001]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900001L4                >
  Elem[002]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900002L4                >
  Elem[003]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900003L4                >
  Elem[004]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900004L4                >
  Elem[005]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900005L4                >
  Elem[006]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900006L4                >
  Elem[007]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900007L4                >
  Elem[008]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900008L4                >
  Elem[009]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900009L4                >
  Elem[010]: tag_val=1 pres_val=1 med_pres=1 med_side=0
               VolumeTag=<900010L4                >
Tag Data for 3.0.0, Element Type MEDIA TRANSPORT:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
Tag Data for 3.0.0, Element Type IMPORT/EXPORT:
  Elem[001]: tag_val=0 pres_val=1 inp_enab=1 exp_enab=1 access=1 full=0 imp_exp=0
  Elem[002]: tag_val=0 pres_val=1 inp_enab=1 exp_enab=1 access=1 full=0 imp_exp=1
```

Figure 108: Output from `sjirdtag`

The `sjirdtag` is a fabulously useful utility if you're using NetWorker with jukeboxes. It reports that current contents of the jukebox *as reported by the jukebox*, not based on the NetWorker cache of the jukebox contents. That's not to say the `nsrjb` output isn't useful – it is, and for the most part, you'll

rely on that. But being able to see what the jukebox is reporting about its own content is very useful when identifying unlabelled volumes or trouble-shooting issues.

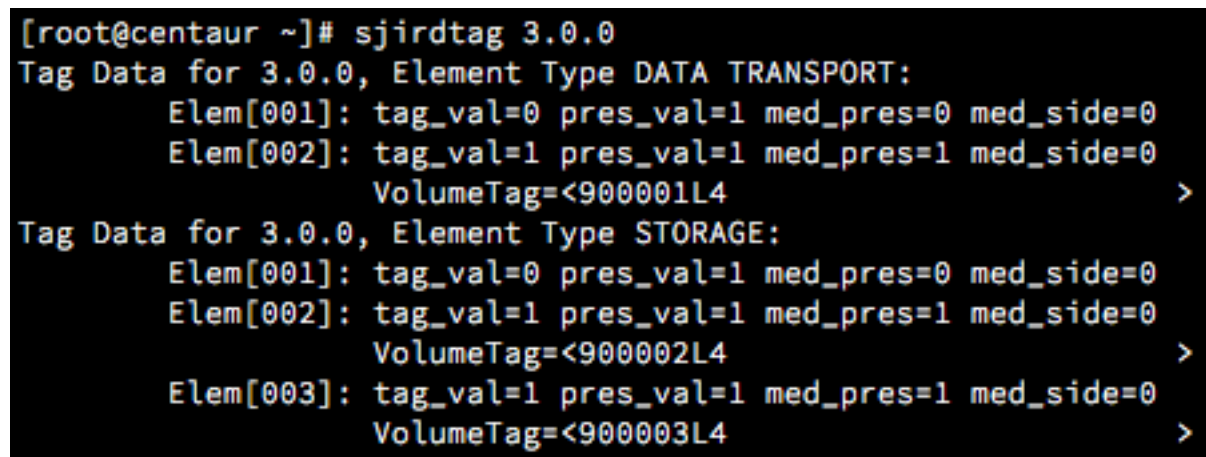
The output for `sjirdtag` is broken up into the following sections:

- **DATA TRANSPORT** – Refers to the tape drives themselves
- **STORAGE** – Refers to the primary slot range
- **MEDIA TRANSPORT** – Refers to the robot arm(s) within the jukebox
- **IMPORT/EXPORT** – Refers to the contents of the CAP/Mail Slot.

For each section, `sjirdtag` will tell you specific details about the component *as well as* media that may be there. For instance, in the output below, neither drive currently has a tape in it:

```
Tag Data for 3.0.0, Element Type DATA TRANSPORT:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[002]: tag_val=0 pres_val=1 med_pres=0 med_side=0
```

However, if we use `nsrjb` to load a tape and then run `sjirdtag` again, we see additional information in the DATA TRANSPORT section:



```
[root@centaur ~]# sjirdtag 3.0.0
Tag Data for 3.0.0, Element Type DATA TRANSPORT:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[002]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900001L4
Tag Data for 3.0.0, Element Type STORAGE:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[002]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900002L4
  Elem[003]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900003L4
```

Figure 109: Noting changes to reported information in `sjirdtag`

You'll see that "Drive Elem[002]" now reports a `med_pres=1` flag, meaning *media is present*, and underneath that line reports the volume tag (barcode) for the volume in the drive – 900001L4. Additionally, if we look at slot 1 in the storage section (Elem[001]) we see the `med_pres=0` flag, meaning the media is no longer present in that slot.

14.2.3 `sjimm`

Sometimes it may be necessary to move media around in a library without using `nsrjb`. It could be that the services are shutdown and you need to quickly get media out, or it could be that you're trying to diagnose whether an issue lays with NetWorker or the tape library itself. In any of these situations, `sjimm` can come to the rescue. The *mm* in the utility name refers to *move media*.

The syntax for `sjimm` is:

```
# sjimm b.t.l source destination
```

Where:

- `b.t.l` is the SCSI bus, target and LUN number of the tape library
- `Source` is the location where the media currently is
- `Destination` is the location where you want the media to go

Both source and destination are defined as:


```
{drive | slot | inlt } number
```

Where the number refers to the reported number via sjirdtag. Drive refers to a tape drive, slot refers to a primary slot number, and inlt refers to the CAP slots.

Two things of note with sjimm:

- sjimm does not handle ejecting media. If you want to transfer media from a drive to a slot, the media needs to be ejected from the drive first;
- If you use sjimm to move media around, you should only do so when NetWorker isn't trying to use the library, *and* you should make sure to reset and reinventory the library in NetWorker after you're complete.

For example, the following command moves a tape from slot 5 in the library at 3.0.0 to slot 2 in the CAP:

```
# sjimm 3.0.0 slot 5 inlt 2
```

Following is an example of the command being run *and* the subsequent sjirdtag output:

```
[root@centaur ~]# sjimm 3.0.0 slot 5 inlt 2
[root@centaur ~]# sjirdtag 3.0.0
Tag Data for 3.0.0, Element Type DATA TRANSPORT:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[002]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900001L4                >
Tag Data for 3.0.0, Element Type STORAGE:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[002]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900002L4                >
  Elem[003]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900003L4                >
  Elem[004]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900004L4                >
  Elem[005]: tag_val=0 pres_val=1 med_pres=0 med_side=0
  Elem[006]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900006L4                >
  Elem[007]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900007L4                >
  Elem[008]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900008L4                >
  Elem[009]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900009L4                >
  Elem[010]: tag_val=1 pres_val=1 med_pres=1 med_side=0
              VolumeTag=<900010L4                >
Tag Data for 3.0.0, Element Type MEDIA TRANSPORT:
  Elem[001]: tag_val=0 pres_val=1 med_pres=0 med_side=0
Tag Data for 3.0.0, Element Type IMPORT/EXPORT:
  Elem[001]: tag_val=0 pres_val=1 inp_enab=1 exp_enab=1 access=1 full=0 imp_exp=0
  Elem[002]: tag_val=1 pres_val=1 inp_enab=1 exp_enab=1 access=1 full=1 imp_exp=0
              VolumeTag=<900005L4                >
```

Figure 110: Using sjimm

15 nsradmin

15.1 Warning

This topic describes activities 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. You should assume **all** examples in this section are prefaced with 'CAUTION – Lab Exercise'.

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 topic, 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.

15.2 Getting Started

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:

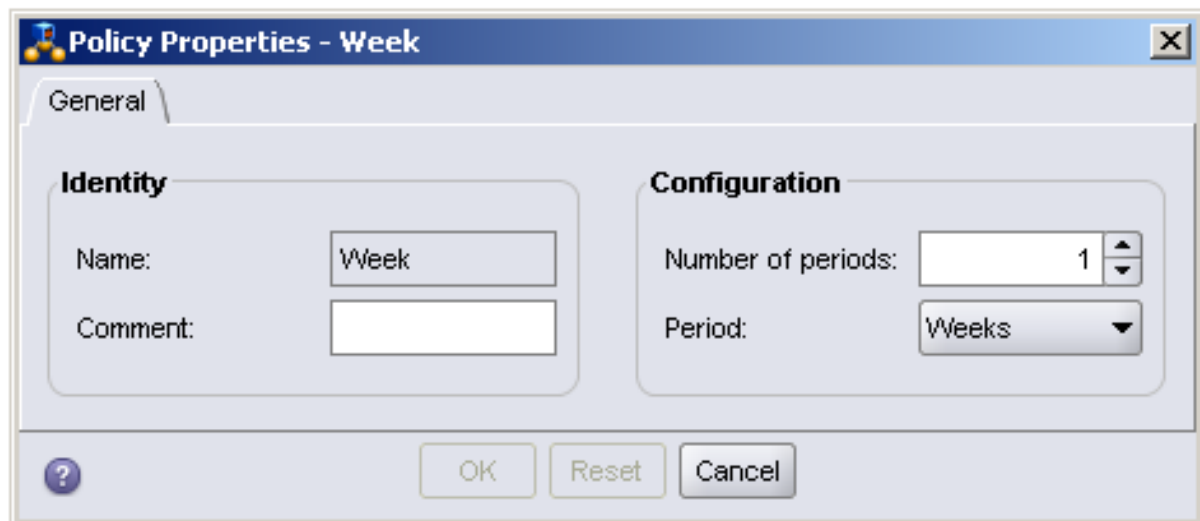


Figure 111: NMC view of a NetWorker resource

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\EMC NetWorker\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, *nsrsla.res* was similarly split up, and organised into a hashed directory structure underneath 'nsrldb' in the same parent directory, 'res'.)

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

```
[root@orilla ~]# ls /nsr/res/nsrdb
00 01 02 03 04 05 06 07 08 09 etrans
[root@orilla ~]# ls /nsr/res/nsrdb/01
0b00cc402f0000001f665c53c0a86404 3d00cc402f0000001f665c53c0a86404
1500cc402f0000001f665c53c0a86404 5100cc402f0000001f665c53c0a86404
1f00cc402f0000001f665c53c0a86404 6500cc402f0000001f665c53c0a86404
290025512e00000027e65c53c0a86404 6f00cc402f0000001f665c53c0a86404
290093082e0000003b456c53c0a86404 8300cc402f0000001f665c53c0a86404
3300cc402f0000001f665c53c0a86404
[root@orilla ~]#
```

Figure 112: NetWorker resource database as files and directories

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 'Week' policy on a server in my lab looks like:

```
comment;;
name: Week;
number of periods: 1;
period: Weeks;
type: NSR policy;
resource identifier: 72.0.204.64.47.0.0.0.31.102.92.83.192.168.100.4(1)
```

Figure 113: NetWorker resource in plain text

It's very important to note here that you **should not**, unless directed to by your support provider, **ever** directly manipulate the actual files within the resource database. 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*.

15.3 Offline vs Online

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

The primary focus for this guide will be *online* mode.

15.4 Your Lab Environment

Throughout this topic, 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 guide will therefore be one where you have:

1. **Installed** the NetWorker server/client/storage node software appropriate to your operating system, downloaded from the EMC Support website, 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:

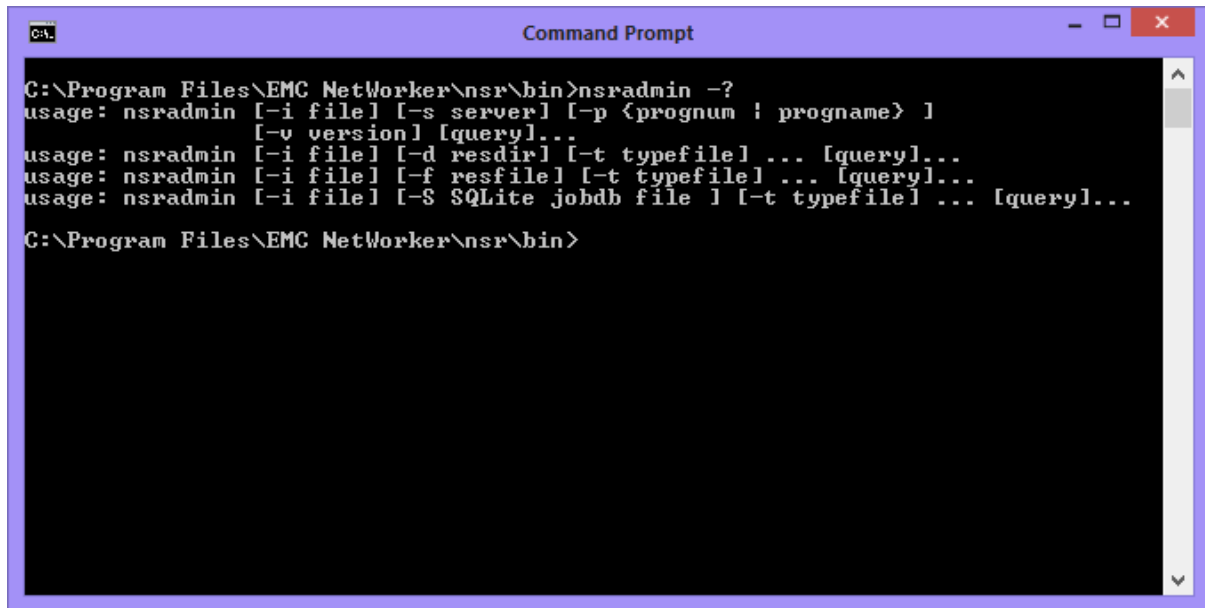
1. A group called “Test”.
2. A client instance for the NetWorker server, with one or two handpicked directories as the save sets. Optimally, you should be looking for a total backup size of between 1 and 3 GB, so that there is enough occupied space to be able to observe backups, but not so much space 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.

15.5 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\EMC NetWorker\nsr\bin”²⁰ (it should however be in the execution path).

To see the usage options for nsradmin, run it with an option of ‘-?’:

²⁰ Older versions of NetWorker had a default install path on Windows of “C:\Program Files\Legato” rather than “C:\Program Files\EMC NetWorker”. If you’ve upgraded from an older to a newer version, the default path is kept.

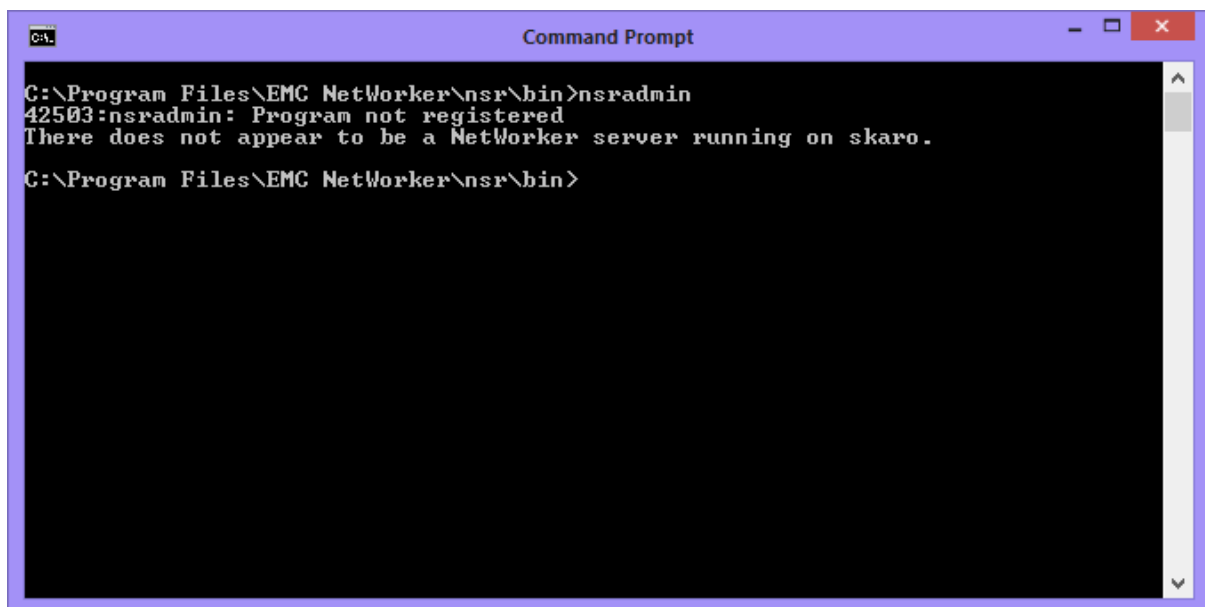


```
C:\Program Files\EMC NetWorker\nsr\bin>nsradmin -?
usage: nsradmin [-i file] [-s server] [-p <prognum : progname> ]
               [-v version] [query]...
usage: nsradmin [-i file] [-d resdir] [-t typefile] ... [query]...
usage: nsradmin [-i file] [-f resfile] [-t typefile] ... [query]...
usage: nsradmin [-i file] [-S SQLite jobdb file ] [-t typefile] ... [query]...

C:\Program Files\EMC NetWorker\nsr\bin>
```

Figure 114: Command line options for nsradmin

If no options are provided to nsradmin, it expects to connect to a NetWorker server running on the current host. Running it on a non-NetWorker server without directing it to either a resource file, directory or NetWorker server will result in the following error:



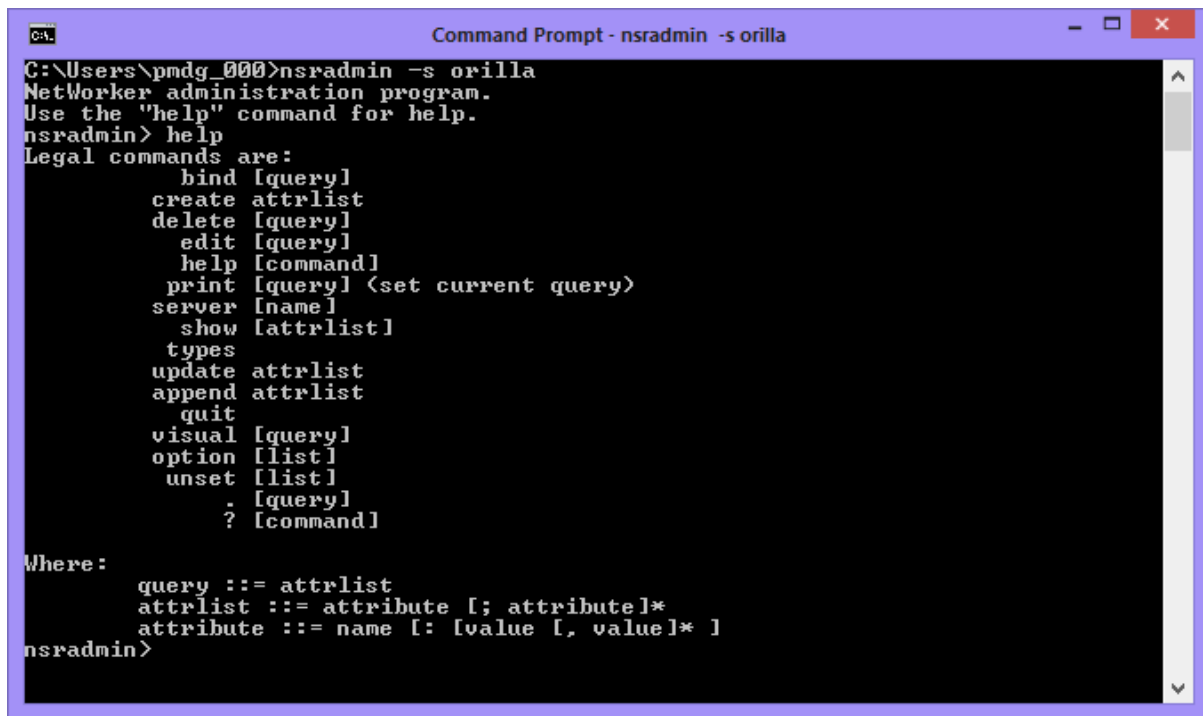
```
C:\Program Files\EMC NetWorker\nsr\bin>nsradmin
42503:nsradmin: Program not registered
There does not appear to be a NetWorker server running on skaro.

C:\Program Files\EMC NetWorker\nsr\bin>
```

Figure 115: Running nsradmin on a client without referencing a server

15.6 Syntax Overview

Like most NetWorker interactive commands, the first keyword to use is *help*. This gives you a brief overview of all the options available in the command. For instance:



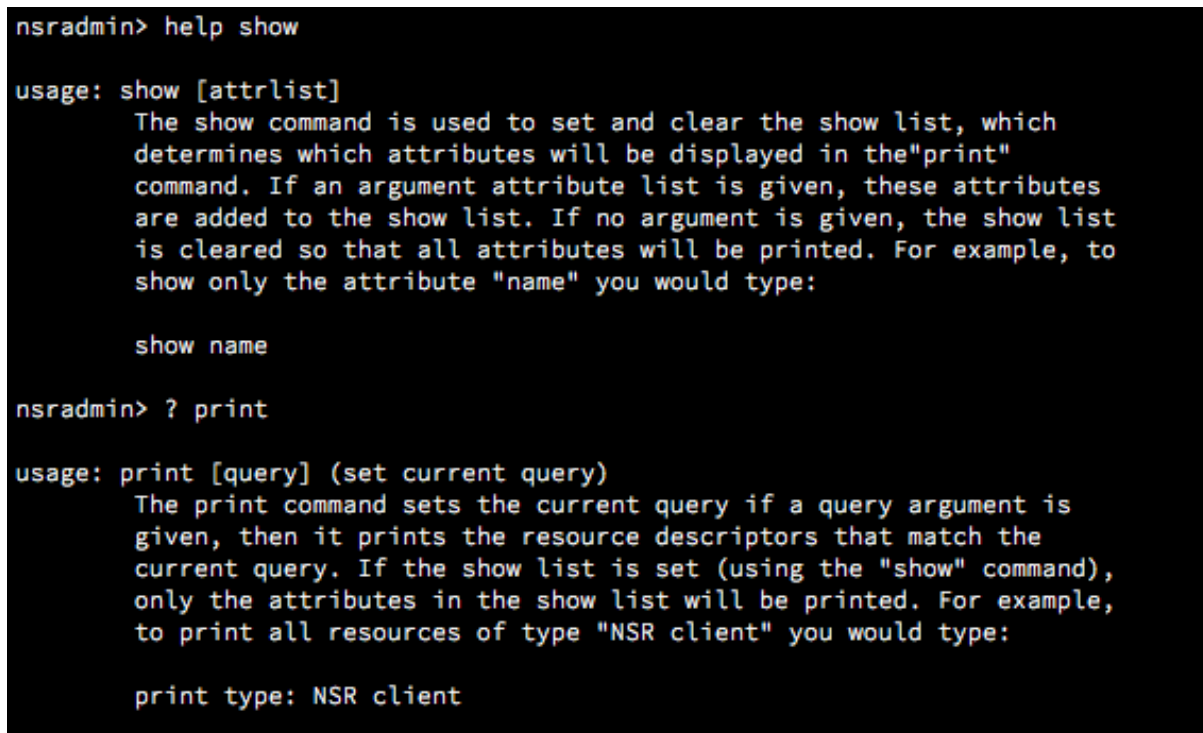
```
C:\Users\pmdg_000>nsradmin -s orilla
NetWorker administration program.
Use the "help" command for help.
nsradmin> help
Legal commands are:
    bind [query]
    create attrlist
    delete [query]
    edit [query]
    help [command]
    print [query] <set current query>
    server [name]
    show [attrlist]
    types
    update attrlist
    append attrlist
    quit
    visual [query]
    option [list]
    unset [list]
    - [query]
    ? [command]

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

Figure 116: Getting help from nsradmin

While almost all actions in nsradmin are the same regardless of whether you're working on Windows or a Unix/Linux platform, it should perhaps be noted that the *edit* and *visual* commands are not available on Windows. To be consistent across operating system types, we'll restrict ourselves to using nsradmin without these options.

Additional information on how individual commands within nsradmin work can be obtained by using either "*help command*" or "*? command*", such as shown below:



```
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
```

Figure 117: Getting help on individual commands with nsradmin

Another good command to use in *nsradmin* is the *types* command, which shows you all the resources that *nsradmin* will allow you to interact with:

```
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> types
Known types: NSR, NSR auditlog, NSR client, NSR device,
              NSR directive, NSR group, NSR label,
              NSR license, NSR lockbox, NSR notification,
              NSR policy, NSR pool, NSR recover,
              NSR Report Home, NSR schedule,
              NSR Snapshot Policy, NSR stage,
              NSR Storage Node, NSR task, NSR usergroup;
nsradmin> _
```

Figure n8: Determining valid configuration types in *nsradmin*

Note that the *types* that are available will change depending on NetWorker version or what program you're communicating with. For instance, *nsradmin* can be invoked against a client program (*nsrexec*) using the command:

```
# nsradmin -p nsrexec -s clientName
```

```
[root@orilla ~]# nsradmin -p nsrexec -s hyperion
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> types
Known types: NSR log, NSR peer information, NSR remote agent,
              NSR system port ranges, NSRLA;
nsradmin> _
```

Figure n9: Viewing types for *nsradmin* against the client program

(An example use of this invocation was outlined in the Reporting section, *nsr_render_log*.)

To exit from *nsradmin*, issue the command *quit*. Returning to *nsradmin* against the Lab backup server itself, we'll look at viewing individual resources. Run the following against your lab server, where *serverName* is the hostname of the lab server:

```
# nsradmin -s serverName
```

You can view resources in *nsradmin* by using the *print* command. This prints *and* sets the current query.

Referring to the help, *nsradmin* tells us:

```
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> help 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
```

Figure 120: nsradmin help for the 'print' command

Global help for nsradmin explains a little more about queries:

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

Thus, to view all policies, we'll be issuing a query that has a single attribute in its attribute list, and that attribute will be a simple *name: value* pair.

The command used will be:

```
nsradmin> print type: NSR policy
```

On a fresh lab server, the output from this command might resemble the following:


```
[root@tara ~]# nsradmin -s tara.pmdg.lab
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> print type: NSR policy
        type: NSR policy;
        name: Month;
        comment: ;
        period: Months;
    number of periods: 1;

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

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

        type: NSR policy;
        name: Quarter;
```

Figure 121: Displaying all policies on a NetWorker server

The output of course, runs over – a default NetWorker server has policies of:

- Day
- Week
- Month
- Quarter
- Year
- Decade

If you want to just quickly see what policies exist, without seeing any details about them, you can use the *show* command to tell nsradmin what you're interested in. The *show* command works as follows:

```
nsradmin> show [attributeNameList]
```

If issued without *any* arguments, it clears any previous restrictions on what is to be shown.

The *attributeNameList*, if specified, will be one or more resource attribute names, semi-colon separated. For preciseness, each name should be terminated by a colon²¹. Thus, if we only wanted to see the *names* of the policies on the NetWorker server, the command would be:

```
nsradmin> show name:
nsradmin> print type: NSR policy
```

If we've stayed in nsradmin, the command might look like the following:

²¹ This is optional, but guarantees precision, and is therefore preferred.

```
nsradmin> show name:
nsradmin> print
name: Month;
name: Week;
name: Year;
name: Quarter;
name: Decade;
name: Day;
```

Figure 122: Viewing just policy names

You'll note that the print command was reduced to *print*, rather than *print type: NSR policy*. There's a reason for that. Remember the help for *print*, which said:

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

For this second command, we're leveraging the *set current query* aspect of the print command. The print command not only displays the output from the query, but stores in nsradmin memory the current query, so that subsequent commands, if desired, can be 'shortcut' to be run against the current command.

If we wanted to limit nsradmin to showing us a single policy – let's say the *Week* policy, then the command becomes:

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

This uses the *attribute list* aspect of the query command. You'll recall an attribute list is a series of one or more attributes, semi-colon separated. Running the command will result in:

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

Figure 123: Viewing a single policy in nsradmin

There's more attributes to a policy than just the name – nsradmin is still honouring the previous *show* command. So, use the command 'show' by itself to turn off the previous restrictions, followed by a 'print' by itself to re-print the last query:

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

Figure 124: Turning off display restrictions and re-printing a query

In addition to the *show* command, which limits the results to particular attributes, there is also the *option* command, which enables features or the display of resources not normally required. You can see what options are available by running the *option* command by itself:

```
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> option

Display options:
    Dynamic: Off;
    Hidden: Off;
    Raw I18N: Off;
    Resource ID: Off;
    Regexp: Off;
nsradmin>
```

Figure 125: nsradmin 'option' command

These features can be enabled or disabled by using the following syntax:

```
nsradmin> option feature
nsradmin> option feature: off
nsradmin> unset feature
```

The first command turns a feature on. The second two are both valid commands for turning the feature *off*. For instance, if we enable the *hidden* feature and print the *year* policy, we see:

```
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
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: Year
      type: NSR policy;
      name: Year;
      comment: ;
      period: Years;
      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;
```

Figure 126: Viewing hidden details of NetWorker resources using the option command

The *hidden* option includes information that you'd normally see in the NetWorker Management Console by enabling *diagnostic* mode.

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 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.

15.7 Starting and Stopping Backups

15.7.1 What you’ll need

In order to complete this section, you’ll need to have configured a test setup as per section 15.4, Your Lab Environment. As a result, you should have:

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

15.7.2 Monitoring

On Unix, Linux and Windows platforms, you can monitor backup activities by running *nsrwatch* in another terminal/DOS session. Alternatively, you can access the *Monitor* tab in the NetWorker Management Console.

15.7.3 Starting a Backup

There are three ways that a group can be started:

- Automatically, as its 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 its 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.

To start a backup within *nsradmin*, you use the following steps:

1. Set the query to specify the group.
2. Change the 'autostart' field to *Start Now*.

You can set the query by either visibly using the *print* (or *delete*) command, or invisibly via the special *dot* command (.). The dot command sets the query without printing any output. I'm not a big fan of this – if you make a mistake in *nsradmin*, you could find yourself needing to recover your bootstrap due to a corrupted configuration. As such, I believe you should be in the habit of *always* setting your query in such a way as you get to see the output of the query²².

You can always use the *show* command first to limit the amount of information that will be printed. Since a group resource is considerably more complex than a time policy, we'll use *show* to limit what we see. The command sequence then would be:

```
# nsradmin
nsradmin> show name;; autostart:
nsradmin> print type: NSR group; name: Test
nsradmin> update autostart: Start Now
```

In this scenario, the process for starting the *Test* group would resemble the following:

²² A few very limited examples in this manual will demonstrate the dot method.

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show name;; autostart:
nsradmin> print type: NSR group; name: Test
                name: Test;
                autostart: Disabled;
nsradmin> update autostart: Start Now
                autostart: Start Now;
Update? y
updated resource id 148.0.252.27.0.0.0.205.77.31.84.192.168.50.7(2)
nsradmin> quit
[root@tara ~]#
```

Figure 127: Starting a group from within nsradmin

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.

If you are going to use the *dot* command instead of the *print* command, limit it to the following scenarios:

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

A note about the autostart attribute:

We started a group by *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.

15.7.4 Stopping a Running Backup

We can equally *stop* a running backup as well. To do this, we want to re-run the Test group, but since no schedule has been defined, there’s a good chance it would do an incremental backup now. To change that and force a full backup each time the group runs, we’re going to set the group’s *level* setting to *full*. The commands for that are:

```
# nsradmin
nsradmin> show name;; level:
nsradmin> print type: NSR group; name: Test
nsradmin> update level: full
```

With output included, the above sequence looks like the following:


```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show name;; level:
nsradmin> print type: NSR group; name: Test
        name: Test;
        level: ;
nsradmin> update level: full
        level: full;
Update? y
updated resource id 148.0.252.27.0.0.0.205.77.31.84.192.168.50.7(17)
```

Figure 128: Setting the level of a group to 'full' always

Now that's been done, we're going to first start the group, wait to see it running in `nsrwatch`, then stop it.

To stop a group, you'll use another of those toggle fields like *autostart*, but this time the field is called *stop now*. The *stop now* field has two potential settings:

- False (default)
- True (used for aborting a running group).

The entire sequence is shown below:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show name;; autostart;; status;; stop now:
nsradmin> print type: NSR group; name: Test
        name: Test;
        autostart: Disabled;
        stop now: False;
        status: idle;
nsradmin> update autostart: Start Now
        autostart: Start Now;
Update? y
updated resource id 148.0.252.27.0.0.0.205.77.31.84.192.168.50.7(18)
nsradmin> print
        name: Test;
        autostart: Disabled;
        stop now: False;
        status: running;
nsradmin> update stop now: True
        stop now: True;
Update? y
updated resource id 148.0.252.27.0.0.0.205.77.31.84.192.168.50.7(21)
nsradmin> █
```

Figure 129: Starting and then stopping a group

You'll notice we used the *show* command here to show four fields:

```
nsradmin> show name;; autostart;; status;; stop now:
```

The *status* field is what is referred to as an *information-only* field. That details the current state of the group (idle, running, cloning, etc.), but you can't directly manipulate that field. Instead, to stop the group, we used the *stop now* field, as shown above.

15.8 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** – A *dynamic* setting that shows save sets that have completed, and their statuses²³.
- **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, "9:index", index;
               status: idle;
nsradmin> █
```

Figure 130: Checking the status of a group

(You'll note that since this server is running 8.2, the 'completion' field does not show up.)

Consider the 'work list' field – this is actually presented as a series of triplets:

- The first is the client the saveset is for
- The second is the saveset level and operation
- The third is the actual saveset

To see this more clearly, restart the group and view the work list once backups are running. This might resemble the following:

```
nsradmin> print type: NSR group; name: Test
               type: NSR group;
               name: Test;
               work list: tara, "full:save", /, tara, "9:index", index;
               status: running;
nsradmin> █
```

Figure 131: Understanding the work list

In this work list, there are 2 pairs of triplets:

- The first:
 - Client 'tara'
 - Level full, operation: 'save'

²³ This may not show correctly under NetWorker 8.2.

- Save set: '/'
- The second:
 - Client 'tara'
 - Level: 9, operation 'index'
 - Save set: 'index'.

15.8.1 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 that 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:

- Set the **clones** attribute to "Yes"
- Set 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 displayed to a minimum – type, status, autostart, clones and clone pool. The process will work as follows:

- Set the show settings 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 you want shown, be sure to issue the "show" command by itself first to clear any previous settings.

The commands to make these changes and then start the group are as follows:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
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
                clone pool: Test Clone;
                clones: Yes;

Update? y
updated resource id 146.0.50.13.0.0.0.0.12.165.152.84.192.168.100.80(15)
nsradmin> update autostart: Start Now
                autostart: Start Now;

Update? y
updated resource id 146.0.50.13.0.0.0.0.12.165.152.84.192.168.100.80(16)
```

Figure 132: Turning cloning on for a group

You can monitor the group in another session using nsrwatch or NMC. Once the group starts cloning, use the 'print' command in nsradmin again to display the new status:

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

Figure 133: Group status while cloning

15.9 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 yet 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.

What we will do is alter the save set settings for our server. Currently they're set to 'All', but we'll reduce them first to a single save set (/usr/share), then increase them using the *append* command. This will resemble the following:

```
nsradmin> show name;; type;; save set:
nsradmin> print type: NSR client; name: backupServer
nsradmin> update save set: /usr/share
...
nsradmin> append save set: /root, /etc
...
nsradmin> print
```

The advantage of *append* should be immediately obvious: if you have a complex or long value already set for a particular field and you want to keep all the existing details, it's both faster and safer to simply append the new value rather than re-typing the entire value.

Obviously, you can't use *append* for all options – you can't for instance *append* another value to a Boolean/selector style field such as the 'autostart' parameter for the group resource. Or more precisely, you can only append values to fields that can take multiple values.

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show name;; type;; save set:
nsradmin> print type: NSR client; name: tara
                type: NSR client;
                name: tara;
                save set: All;
nsradmin> update save set: /usr/share
                save set: /usr/share;
Update? y
updated resource id 93.0.50.13.0.0.0.0.12.165.152.84.192.168.100.80(8)
nsradmin> append save set: /root, /etc
                save set: /root, /etc;
Append? y
updated resource id 93.0.50.13.0.0.0.0.12.165.152.84.192.168.100.80(9)
nsradmin> print
                type: NSR client;
                name: tara;
                save set: /usr/share, /root, /etc;
```

Figure 134: Using append instead of update

In this case, Windows will be slightly different – within nsradmin, it's necessary to *escape* backslashes by using a preceding backslash, and keep the paths in quotes (due to the colon). Thus, if a Windows client already had a save set of *C:\Temp* and we wanted to add *C:\Documents and Settings*, the sequence might resemble the following:

```
nsradmin> show name;; save set:
nsradmin> print type: NSR client; name: clientName
...
nsradmin> append save set: "C:\\Documents and Settings"
...
```

Using a client called 'faraway', this sequence would be as follows:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show name;; group;; save set:
nsradmin> print type: NSR client; name: faraway
                name: faraway;
                group: Windows;
                save set: "C:\\Temp";
nsradmin> append save set: "C:\\Documents and Settings"
                save set: "C:\\Documents and Settings";
Append? yes
updated resource id 37.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(4)
```

Figure 135: Specifying Windows save sets in nsradmin

15.10 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 NetWorker Management Console for any of these, we'll do the complete setup within *nsradmin*.

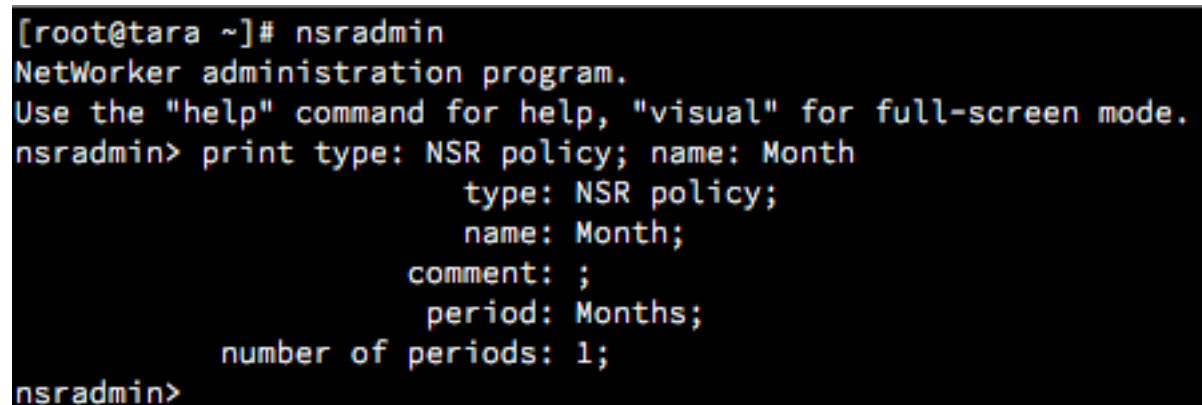
15.10.1 Browse and Retention Policies

As you'd know from using NMC, a policy is neither a browse, nor a retention policy, until it is actually assigned to a client, group or pool – until then it's nothing more than a simple definition of time.

We'll create "Daily" and "Monthly" policies, with the details being as follows:

- Daily – A period of 5 weeks
- Monthly – 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 know from previous sections there is a 'Month' policy, so let's look at that again:



```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> print type: NSR policy; name: Month
                type: NSR policy;
                name: Month;
                comment: ;
                period: Months;
                number of periods: 1;
nsradmin>
```

Figure 136: The 'Month' policy

This tells us the attributes we'll need to set are:

- type
- name
- period
- number of periods

If you're unsure of what the period *types* are that you can use, you can view all the current, in-use period types using the following command:

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

On our server, the output will resemble the following:


```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> show period:
nsradmin> print type: NSR policy
                period: Years;

                period: Months;

                period: Days;

                period: Years;

                period: Weeks;

                period: Months;
nsradmin>
```

Figure 137: Viewing the different period types available to NetWorker policies

Obviously the effectiveness of this technique will be dependent on how many resources there are of a particular type. When there are quite a few resources, it's usually more effective to resort to the command reference guides. For Unix and Linux, you can view the documentation for any resource type by running "man nsr_type" – e.g., in this case, "man nsr_policy". (These manual pages are included in the NetWorker command reference guide, and thus available for all platforms.)

We want to create a Monthly policy that gives us a time period of 13 months, as a Daily policy that gives us a time period of 5 weeks. The commands will therefore be:

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

When executed, this will appear as follows:

```
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 39.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> create type: NSR policy; name: Monthly; period: Months; number of peri
ods: 13
                type: NSR policy;
                name: Monthly;
                period: Months;
                number of periods: 13;
Create? y
created resource id 40.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 138: Creating Daily and Monthly policies

Note:

- If a command is longer than a line, you can either let it naturally flow onto the following line or ensure you press enter after using a value and ending the line with a semi-colon.

To verify the policies now exist, use the *print* command:

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

Figure 139: Viewing the newly created Daily and Monthly policies

15.10.2 Schedules

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

- **Daily** – Fulls on Friday, incrementals Saturday to Thursday, and the last Friday of the month skipped;
- **Monthly** – Skips every day of the month, except for the last Friday of the month, where it does a full backup.

If you've only worked with schedules in NMC, they'll look a little different in nsradmin. Consider the *Default* schedule:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> print type: NSR schedule; name: Default
                type: NSR schedule;
                name: Default;
                comment: ;
                period: Week;
                action: full incr incr incr incr incr incr;
                override: ;
nsradmin>
```

Figure 140: Settings for the Default schedule

The three key attributes of a schedule (other than the name and type) are:

- **period** – This specifies the schedule period. It will be either *week* or *month*. This in turn defines how NetWorker will interpret the *action* attribute.
- **action** – A list of levels to be performed on consecutive days. For a *week* period, this action list covers Sunday through Saturday, *in that order*. For a month schedule, the action list covers the first through to the thirty-first, in that order.
- **override** – Any special changes to the backup to suit particular dates or special days.

At this point it may look like creating schedules is easier in the GUI – but nsradmin has a few shortcuts up its sleeve. Let's first consider the Daily schedule, where we want:

- Full backups on Friday
- Incremental backups all other days of the week
- Skip the backup on the final Friday of every month.

The first two requirements translate to attributes as follows:

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

There's already one nsradmin short-cut we can take in the above details: nsradmin doesn't need the full length description of each level. Therefore, the action list could be shortened to "i i i i i f i". However, that's not the main short-cut.

In the standard calendar view of a schedule in NMC, an override to skip the last Friday of every month would necessitate:

- Select the schedule, then, right-click the last Friday, go into the 'override' drill down menu and choose 'skip'.
- Repeat for as many months as necessary (or until you get tired of it)
- Setting a calendar reminder to extend the skips from the month you stopped at

However, nsradmin, like the non-calendar view of NMC, allows you to enter a "set once" style override, namely:

- **override** – "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
```

Executing this will work as follows:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
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
created resource id 41.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 141: Creating the Daily schedule

Moving on to the Monthly schedule, we'll see the other shortcut offered by nsradmin. You'll recall our goal is:

- 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, create an action list of "skip skip skip skip ... skip" that has 31 entries in it. We already know we can shorten level names, so this could at least drop down to 31 x "s": "s s s s s s s s s ... s".

Another way to go about it is to treat the schedule like a *Weekly* one (since the override will accomplish everything we want), and then just use a 7-day action list: "s s s s s s s". But, that's still not the easiest way of doing it.

Earlier I said the action list for a month is 31 days long – that means if you have a Month based schedule with 31 entries in it and it is applied to February, all those extra days after the 28th or the 29th (if it's a leap year) are ignored. Equally however, the action list supports being *shorter* than the time period (week or month). When this is the case, the action list is looped by NetWorker to cover any 'missing' days in the list. Thus, our Monthly schedule can be created as follows:

```
nsradmin> create type: NSR schedule; name: Monthly;
period: Month; action: s;
override: full last Friday every month
```

Here's what it looks like being run:

```
[root@tara ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
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 42.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 142: Creating the Monthly schedule

If you look in NMC, you'll see these schedules have been correctly accepted:

Schedule Properties - Daily

Name:

Period:

Comment:

December 2014 31 December 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
	i	i	i	i	f	i
7	8	9	10	11	12	13
i	i	i	i	i	f	i
14	15	16	17	18	19	20
i	i	i	i	i	f	i
21	22	23	24	25	26	27
i	i	i	i	i	*Skip	i
28	29	30	31			
i	i	i	i			

? OK Reset Cancel

Figure 143: Daily schedule as shown by NMC

Schedule Properties - Monthly

Name:

Period:

Comment:

December 2014 31 December 2014

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
		s	s	s	s	s
7	8	9	10	11	12	13
	s	s	s	s	s	s
14	15	16	17	18	19	20
	s	s	s	s	s	s
21	22	23	24	25	26	27
	s	s	s	s	*Full	s
28	29	30	31			
	s	s	s			

? OK Reset Cancel

Figure 144: Monthly schedule as shown by NMC

15.10.3 Groups

Now the schedules and time policies have been created, we can move on to create the Daily and Monthly groups that we'll use to execute our backups with.

As always, when creating a new resource, it's a good idea to see what options are available using an existing one – in this case we'd use the *Default* group:


```
nsradmin> print type: NSR group; name: Default
          type: NSR group;
          name: Default;
          comment: ;
          snapshot: False;
          autostart: Disabled;
          autorestart: Disabled;
          client subset: ;
          start time: "21:00";
          last start: ;
          last end: ;
          next start: Disabled;
          interval: "24:00";
          restart window: "12:00";
          force incremental: Yes;
          savegrp parallelism: 0;
          client retries: 1;
          client retry delay: 0;
          timestamp: none;
          clones: No;
          clone mode: Start on group completion;
          clone pool: Default Clone;
          success threshold: Warning;
          options: Revert to full when synthetic full fails,
```

Figure 145: Viewing the Default group

A group is significantly more complex in options than either a policy or a schedule, so it won't fit within a standard window length. Run the command yourself and view output carefully:

```
nsradmin> print type: NSR group; name: Default
```

When we create a new group, we're going to make use of the following attributes:

- **type** – NSR group
- **name** – Daily
- **autostart** – Enabled
- **start time** – "21:35"
- **schedule** – Daily
- **browse policy** – Daily
- **retention policy** – Daily

By setting the schedule to Daily, we ensure that all clients added to this group are backed up with the same schedule, which works in *most* circumstances. Setting the browse and retention policy to *Daily* means all backups executed by the group have those browse and retention policies applied²⁴.

Our create command for the Daily group will therefore be:

```
nsradmin> create type: NSR group; name: Daily;
autostart: Enabled; start time: "21:35"; schedule: Daily;
browse policy: Daily; retention policy: Daily
```

Within nsradmin, the command will execute as follows:

²⁴ Almost all. NMC database backups don't inherit this feature.

```
nsradmin> create type: NSR group; name: Daily;
autostart: Enabled; start time: "21:35"; schedule: Daily;
browse policy: Daily; retention policy: Daily
        type: NSR group;
        name: Daily;
        autostart: Enabled;
        start time: "21:35";
        schedule: Daily;
        browse policy: Daily;
        retention policy: Daily;
Create? y
created resource id 44.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 146: Creating the Daily group

Our create command for the Monthly group will be quite similar, except we'll be swapping the key word 'Daily' for 'Monthly' in every occurrence. We'll also change the start time slightly, because no two groups should start at the same time in NetWorker:

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

When executed, this will resemble the following:

```
nsradmin> create type: NSR group; name: Monthly;
autostart: Enabled; start time: "21:40"; schedule: Monthly;
browse policy: Monthly; retention policy: Monthly
        type: NSR group;
        name: Monthly;
        autostart: Enabled;
        start time: "21:40";
        schedule: Monthly;
        browse policy: Monthly;
        retention policy: Monthly;
Create? y
created resource id 45.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 147: Creating the Monthly group

15.10.4 Clients

So far we've created policies, schedules and groups, and now it's time for the clients. We won't actually be backing anything up, so rather than worrying about allocating test machines, we'll just create a couple of dummy hosts file 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 there's a good chance it won't interfere with anything else, but check before setting up your hosts file entries that the IP addresses chosen (and the hostnames) don't appear on your network or in DNS respectively.

On a Linux or Unix NetWorker server, you can add those entries into `/etc/hosts`. On a Windows NetWorker server, you'd add the entries to `C:\windows\system32\drivers\etc\hosts`. (On both, it's usually a good idea to restart the NetWorker services after manually adding hosts entries.)

Once the hosts file entries have been created, we can create the client entries. We'll create one entry for each client, giving each entry the *longer* browse and retention policy (since these fields can't be blank). We'll assign browse and retention policies at the group level, and we'll also assign the retention policy at the pool level (which will catch 'errant' save sets like the NMC database).

As per previous resources, we'll start by looking at an existing client – in this case, the backup server itself, which was placed in the 'Test' group earlier:

```
nsradmin> print type: NSR client; group: Test
               type: NSR client;
               name: tara;
               server: tara;
               client id: \
81fe9e2d-00000004-5498a50e-5498a50d-00015000-3c48a956;
               scheduled backup: Enabled;
               comment: ;
               Save operations: ;
               archive services: Disabled;
               schedule: Default;
               browse policy: Month;
               retention policy: Year;
               statistics: elapsed = 240856, index size (KB) = 14426,
                           amount used (KB) = 14426, entries = 94922;
               directive: ;
               group: Test;
               save set: /usr/share, /root, /etc;
Backup renamed directories: Enabled;
Checkpoint enabled: Disabled;
Checkpoint granularity: Directory;
Parallel save streams per save set: Disabled;
```

Figure 148: Viewing an existing client instance

While there are a lot of potential attributes you can use for creating a client, we'll be limiting ourselves to the following attributes:

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

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
```

Executed in nsradmin, this will appear as follows:

```
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.ab
        type: NSR client;
        name: test1;
        browse policy: Monthly;
        retention policy: Monthly;
        group: Daily, Monthly;
        save set: All;
        aliases: test1, test1.my.ab;
        parallelism: 1;
Create? y
created resource id 46.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 149: Creating a new client in nsradmin

You'll note there's a typo in the above – the aliases were entered incorrectly²⁵. We can fix this by printing the client then changing the aliases:

```
nsradmin> print type: NSR client; name: test1
...
nsradmin> update aliases: test1, test1.my.lab
```

The second client creation command will be quite similar to first, though this time without the typo:

```
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 47.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 150: Second client create command in nsradmin

15.10.5 Pools

The final configuration components to be setup are the pools. We'll define four pools – Daily, Daily Clone, Monthly and Monthly Clone. We won't be backing up to these pools, so there won't be any need for additional advanced file type devices or media.

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

²⁵ You'll note that NetWorker allowed an alias used that by rights was incorrect. NetWorker doesn't do name resolution checking on aliases, as aliases may refer to private names defined only on the client.

```
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;
           media type required: ;
           volume type preference: ;
           max parallelism: 0;
           mount class: default;
           WORM pool: No;
           create DLTWORM: No;
           barcode prefix: ;
```

Figure 151: Viewing the Default pool in nsradmin

While there are a variety of attributes available for pools, we'll focus on just a few particular ones, namely:

- 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 for a backup or a clone job, 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 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, confirming the media is readable.)

The “store index entries” attribute is set to “Yes” by default, which is appropriate for backup pools. However, NetWorker requires this setting to be set to “No” for Backup Clone pools.

The create command for the Daily pool will be as follows:

```
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
```

When executed in nsradmin, this will look like the following:

```
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 48.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 152: Creating the Daily pool in nsradmin

Note:

If you get an alert about needing to create a label template first, or to re-create the resource in order to automatically create a label template: it’s time to upgrade to a newer version of NetWorker.

The command to create the **Daily Clone** pool will be similar, except we won’t specify any groups, and we *will* include the store index entries attribute. The command will resemble the following:

```
nsradmin> 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
```

When executed in nsradmin, this will look like the following:


```
nsradmin> 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
        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 49.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 153: Creating the Daily Clone pool

The Monthly and Monthly Clone pools will be almost exactly the same as the dailies, just replacing 'Daily' for 'Monthly' in every occurrence:

```
nsradmin> create type: NSR pool; name: Monthly; enabled: Yes;
pool type: Backup; auto media verify: Yes; recycle to other pools: Yes;
recycle from other pools: Yes; retention policy: Monthly;
groups: 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 50.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 154: Monthly pool

```
nsradmin> 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 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 51.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 155: Monthly Clone pool

That's it for the configuration we want to create, but there is one thing left to be done – like any good backup environment, we'll want our configuration to automatically clone. Now the pools have been created, we can finally make those changes.

15.10.6 Revisiting the Groups

We had previously configured the Test group to clone to the Test Clone pool, and we'll use the same procedure to change the Daily group to clone to the Daily pool, and the Monthly group to clone to the Monthly pool. These commands will be as follows:

```
nsradmin> show name;; clones;; clone pool:
nsradmin> print type: NSR group; name: Daily
...
nsradmin> update clones: Yes; clone pool: Daily Clone
...
nsradmin> print type: NSR group; name: Monthly
...
nsradmin> update clones: Yes; clone pool: Monthly Clone
...
```

When executed in nsradmin, this will resemble the following:

```
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 44.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(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 45.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(2)
```

Figure 156: Configuring the Daily and Monthly groups to clone

15.11 Monitoring Devices

Another basic activity you can perform with nsradmin is monitoring devices while activities are running. This is relatively easy, and since it can be easily scripted, handy for both on-the-spot checks and performance analysis.

Let's look at a device:

```
nsradmin> print type: NSR device; name: Backup
               type: NSR device;
               name: Backup;
               comment: ;
               description: ;
device access information: /aftd/backup;
  enable fibre channel: No;
  fibre channel hostname: ;
    message_I18N: "writing, done ";
    message: "writing, done ";
    volume name: Backup.01;
    media family: disk;
    media type: adv_file;
    enabled: Yes;
    read only: No;
    target sessions: 4;
    max sessions: 32;
    max nsrmmmd count: 12;
  verify label on eject: No;
    parent jukebox: ;
    cleaning required: No;
```

Figure 157: Viewing a device in nsradmin

Note:

If you're using NetWorker 7.6.x or lower, AFTD devices are named based on their directory path – for instance, the device shown above, under 7.6.x would be called “/aftd/backup”²⁶.

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

- message
- message_I18N

What we'll do is to start the Test group from before, and monitor what the devices do, both on backup, then clone. To do this, we'll step through the following sequence:

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 backup.

This will resemble the following:

²⁶ But you really should upgrade to an 8.x release as soon as possible.

```
nsradmin> . type: NSR group; name: Test
Current query set
nsradmin> update autostart: Start Now
                autostart: Start Now;
Update? y
updated resource id 146.0.50.13.0.0.0.12.165.152.84.192.168.100.80(33)
nsradmin> show name;; message;; message_I18N
nsradmin> print type: NSR device
                name: Backup;
                message_I18N: "writing, 152 MB, 3 sessions";
                message: "writing, 152 MB, 3 sessions";

                name: Clone;
                message_I18N: space recovered from volume Clone.01;
                message: space recovered from volume Clone.01;
nsradmin> print
                name: Backup;
                message_I18N: "writing at 5585 KB/s, 1265 MB";
                message: "writing at 5585 KB/s, 1265 MB";

                name: Clone;
                message_I18N: space recovered from volume Clone.01;
                message: space recovered from volume Clone.01;
nsradmin> print
                name: Backup;
                message_I18N: "reading, data ";
                message: "reading, data ";

                name: Clone;
                message_I18N: "writing at 163 MB/s, 420 MB";
                message: "writing at 163 MB/s, 420 MB";
```

Figure 158: Viewing active devices using nsradmin

Note – if you’re working in an English-language only environment, you can choose to leave out the *message_I18N* field.

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.

15.12 Deleting Resources

For the last of the basic exercises with nsradmin, we’re going to delete the Daily and Monthly setup. There are two ways that deletions can be done:

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

Almost all of the time you should only perform *delete* operations using the second method. Particularly if you’re doing any *scripted* deletes, you should definitely only use the second method. With that in mind, all delete exercises will be of the form *delete query*.

As you might have noticed when using NMC, 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 parts of the configuration.

The first set of resources we can delete are the clients – these aren't referenced anywhere else, so it's safe to get rid of them with the following command:

```
nsradmin> show name;; save set;; group:
nsradmin> delete type: NSR client; group: Daily
```

You'll note in the above that we're not specifying each client individually. Because we want to undo everything we've done, we can shortcut the client deletions by deleting all clients in the *Daily* group:

```
nsradmin> show name;; save set;; group:
nsradmin> delete type: NSR client; group: Daily
                name: test2;
                group: Daily, Monthly;
                save set: All;

Delete? y
deleted resource id 47.0.159.55.0.0.0.131.131.159.84.192.168.100.80(1)
                name: test1;
                group: Daily, Monthly;
                save set: All;

Delete? y
deleted resource id 46.0.159.55.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 159: Deleting clients using nsradmin

Moving on, we'll first change our groups *Daily* and *Monthly* to not use the clone pools, then we can delete the groups followed by the pools. We'll use a previously not shown option to clear the clone pool setting. This will be done as follows:

```
nsradmin> show name;; clones;; clone pool:
nsradmin> print type: NSR group; name: Daily
...
nsradmin> update clones: No; clone pool:
nsradmin> . type: NSR group; name: Monthly
nsradmin> update clones: No; clone pool:
```

Note for the first group I used the *print* command, but for the second group I used the *set* (.) command. This was merely to demonstrate the interchangeability of the commands.

When executed in nsradmin, this will resemble the following:


```
nsradmin> show name;; clones;; clone pool:
nsradmin> print type: NSR group; name: Daily
                name: Daily;
                clones: Yes;
                clone pool: Daily Clone;
nsradmin> update clones: No; clone pool:
                clone pool: ;
                clones: No;

Update? y
updated resource id 44.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(3)
nsradmin> print type: NSR group; name: Monthly
                name: Monthly;
                clones: Yes;
                clone pool: Monthly Clone;
nsradmin> update clones: No; clone pool:
                clone pool: ;
                clones: No;

Update? y
updated resource id 45.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(3)
```

Figure 160: Turning cloning off for groups

With the clone pools removed from the groups, we can now go on and remove the pools. Note that if we had actually generated *backups* or *clones* to these pools, it would be necessary to delete the volumes containing those backups/clones prior to deleting the pool resources.

The commands to delete the pools will be:

```
nsradmin> show name:
nsradmin> delete type: NSR pool; name: Daily Clone
nsradmin> delete type: NSR pool; name: Monthly Clone
nsradmin> delete type: NSR pool; name: Daily
nsradmin> delete type: NSR pool; name: Monthly
```

Of course, using the *show* command before a delete isn't required, but it makes it a good way of reducing the clutter/superfluous information on-screen and allowing you to more readily see and confirm what you're deleting.

In nsradmin, this sequence would appear as follows:

```
nsradmin> show name:
nsradmin> delete type: NSR pool; name: Daily Clone
                name: Daily Clone;
Delete? y
deleted resource id 49.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> delete type: NSR pool; name: Monthly Clone
                name: Monthly Clone;
Delete? y
deleted resource id 51.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> delete type: NSR pool; name: Daily
                name: Daily;
Delete? y
deleted resource id 48.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> delete type: NSR pool; name: Monthly
                name: Monthly;
Delete? y
deleted resource id 50.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 161: Deleting pools in nsradmin

With the pools deleted, we'll move on to delete the groups, schedules and policies:

```
nsradmin> show name;; type:
nsradmin> delete type: NSR group; name: Daily
nsradmin> delete type: NSR group; name: Monthly
nsradmin> delete type: NSR schedule; name: Daily
nsradmin> delete type: NSR schedule; name: Monthly
nsradmin> delete type: NSR policy; name: Daily
nsradmin> delete type: NSR policy; name: Monthly
```

This entire sequence will resemble the following when executed in nsradmin:

```
nsradmin> show name;; type:
nsradmin> delete type: NSR group; name: Daily
                        type: NSR group;
                        name: Daily;

Delete? y
deleted resource id 44.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(5)
nsradmin> delete type: NSR group; name: Monthly
                        type: NSR group;
                        name: Monthly;

Delete? y
deleted resource id 45.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(5)
nsradmin> delete type: NSR schedule; name: Daily
                        type: NSR schedule;
                        name: Daily;

Delete? y
deleted resource id 41.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> delete type: NSR schedule; name: Monthly
                        type: NSR schedule;
                        name: Monthly;

Delete? y
deleted resource id 42.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> delete type: NSR policy; name: Daily
                        type: NSR policy;
                        name: Daily;

Delete? y
deleted resource id 39.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
nsradmin> delete type: NSR policy; name: Monthly
                        type: NSR policy;
                        name: Monthly;

Delete? y
deleted resource id 40.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 162: Deleting the groups, schedules and policies

That ends the basic coverage of nsradmin. The remaining topics are now going to introduce the real power of nsradmin – using it non-interactively and in scripting.

15.13 Bulk Commands

Consider the scenario where say, 20 new servers were going to be added to the environment. They'll all be standard build systems without databases, which means using either NMC or nsradmin in interactive mode will be tedious to add the systems.

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

In order to demonstrate this, we'll need to extend our previous 2 entries in the hosts file, adding another 18. Previously we established hosts entries of:

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

Now, add another 18 test host entries, so this section of your hosts file reads:

```
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 scenario, we'd already have groups, schedules, pools, etc., configured. Instead of going through and setting all these up, we'll just specify the name, aliases and parallelism, and have NetWorker fill in all the other details for us.

Next, create a text file that has the following entries:

```
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”. Ensure when saving the file that there is a blank line at the bottom of the file – if the file ends on the same line as a command, nsradmin may not execute the final command. (Note that the extension is optional, but it’s best to pick an extension and stick with it.)

We’re going to get nsradmin to run all the commands in that file without going into interactive mode. If you’re on Linux or Unix, you can run it as:

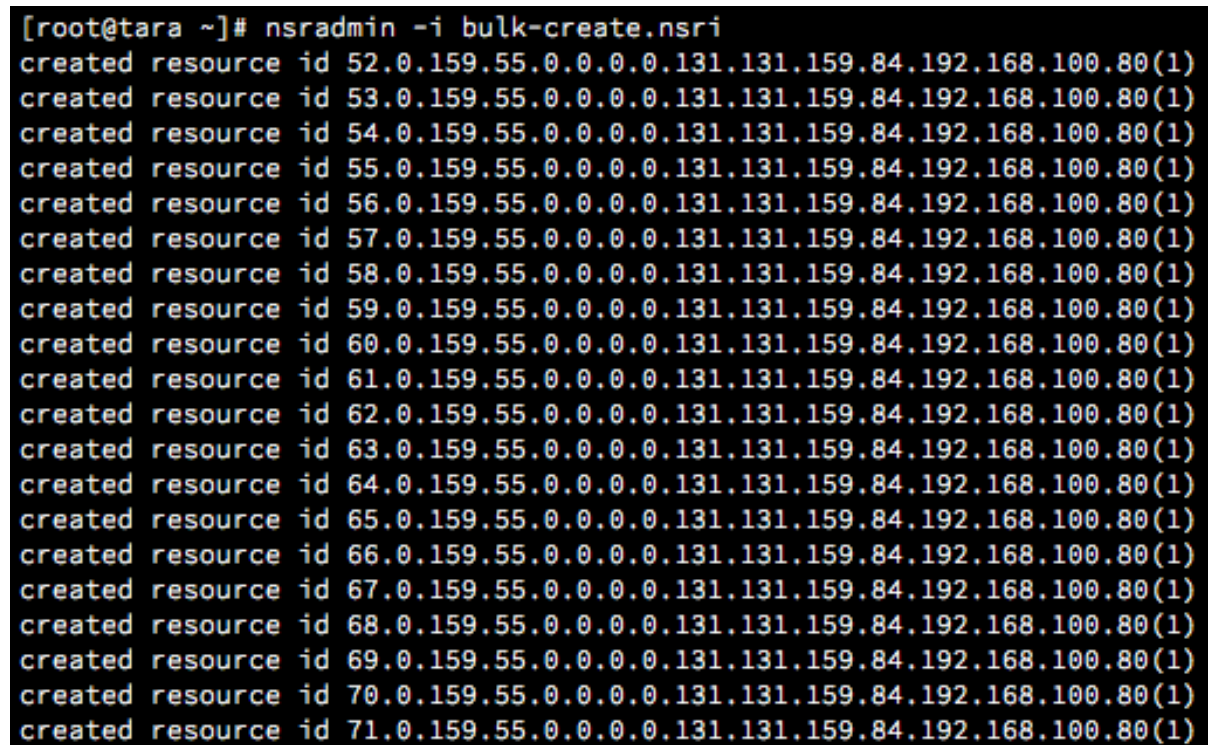
```
# nsradmin -i file
```

Where file is the name of the *file* you’ve created.

On Windows systems, you may be able to run it as simply as the above – or else, you might have to call the full path to the file, i.e.:

```
C:\> nsradmin -i X:\Path\to\file
```

Assuming we’ve saved *bulk-create.nsri* in the current working directory, the execution would appear as follows:



```
[root@tara ~]# nsradmin -i bulk-create.nsri  
created resource id 52.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 53.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 54.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 55.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 56.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 57.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 58.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 59.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 60.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 61.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 62.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 63.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 64.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 65.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 66.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 67.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 68.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 69.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 70.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)  
created resource id 71.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 163: Bulk addition of clients to NetWorker using nsradmin

All commands you use in nsradmin interactively can be used non-interactively. For instance, if you wanted to remove all those clients from NetWorker, you’d create a script file (say, bulk-delete.nsri) with the following content:

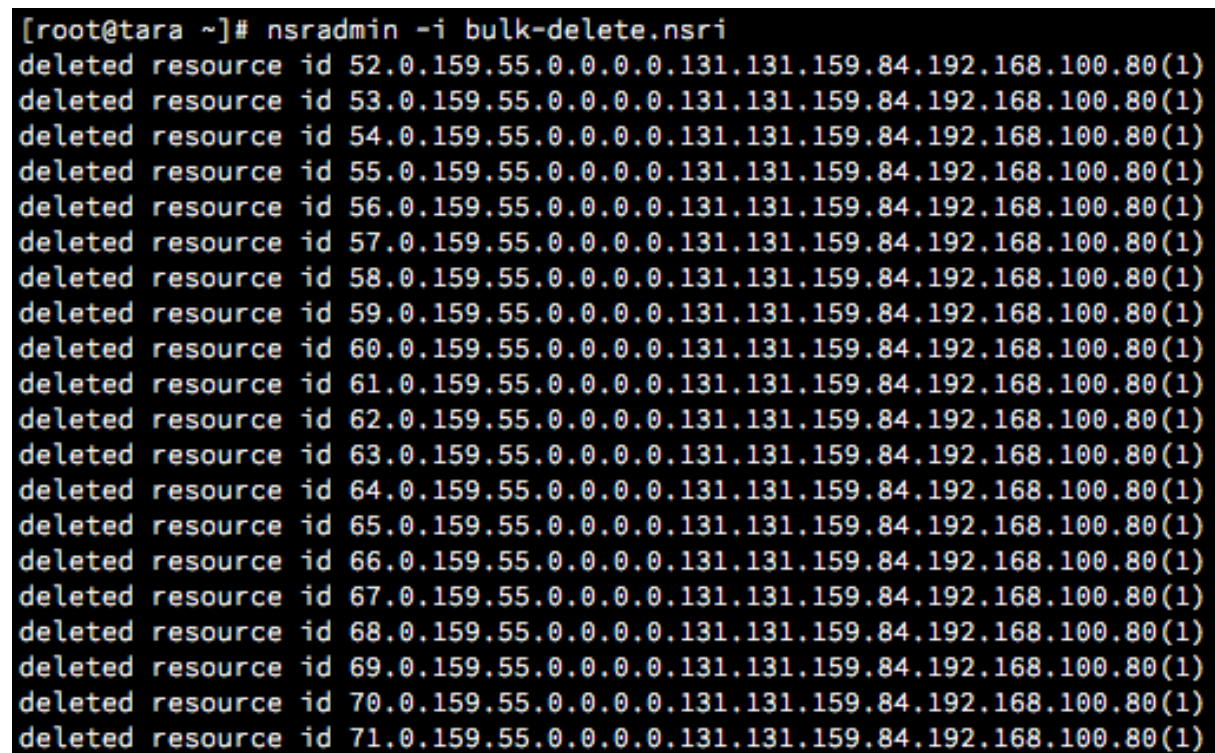
```
delete type: NSR client; name: test1  
delete type: NSR client; name: test2  
delete type: NSR client; name: test3  
delete type: NSR client; name: test4  
delete type: NSR client; name: test5  
delete type: NSR client; name: test6  
delete type: NSR client; name: test7  
delete type: NSR client; 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
```

This could then be run as:

```
# nsradmin -i bulk-delete.nsri
```

Output from the delete would be similar to the following:

A screenshot of a terminal window showing the output of the command 'nsradmin -i bulk-delete.nsri'. The output consists of 13 lines, each starting with 'deleted resource id' followed by a long alphanumeric string and a number in parentheses. The strings appear to be IP addresses or identifiers. The terminal has a dark background with light-colored text.

```
[root@tara ~]# nsradmin -i bulk-delete.nsri
deleted resource id 52.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 53.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 54.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 55.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 56.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 57.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 58.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 59.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 60.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 61.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 62.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 63.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 64.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 65.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 66.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 67.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 68.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 69.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 70.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
deleted resource id 71.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
```

Figure 164: Performing a bulk delete in nsradmin

15.14 Scripting

Consider a scenario where you want support/operations staff responsible for setting up new servers in the backup environment to have simple commands they can run to interactively fill in the details required for new clients within NetWorker. This will require scripting.

Note:

- For the examples used in this section, there'll be no input validation. For more comprehensive scripting of nsradmin, you'd be advised to use a good scripting language such as Perl or Python, and perform appropriate input validation to ensure the values you feed through to nsradmin are acceptable.

15.14.1 Introductory Scripting

A less ambitious starting point will get us going with scripting. Consider a scenario where 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.

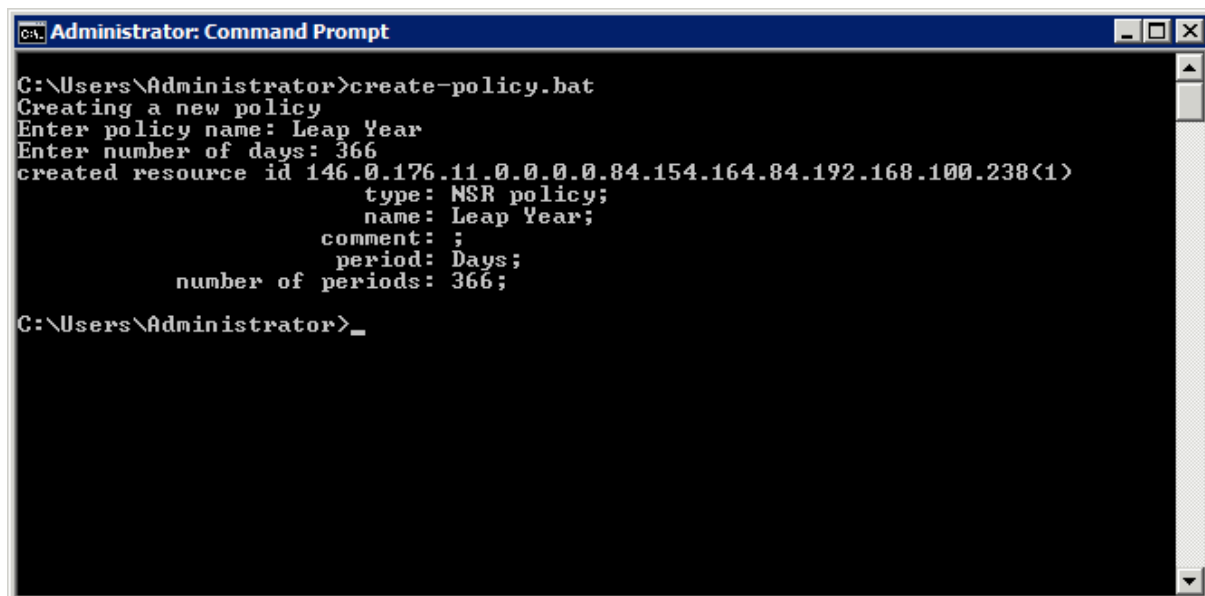
On Windows, this might be called “create-policy.bat” and it would have the following content:

```
@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:



```
C:\Users\Administrator>create-policy.bat
Creating a new policy
Enter policy name: Leap Year
Enter number of days: 366
created resource id 146.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
      type: NSR policy;
      name: Leap Year;
      comment: ;
      period: Days;
      number of periods: 366;
C:\Users\Administrator>_
```

Figure 165: Running the create-policy.bat script

On Unix/Linux systems, using Perl we could do a similar function with the following script, called “create-policy.pl”:

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

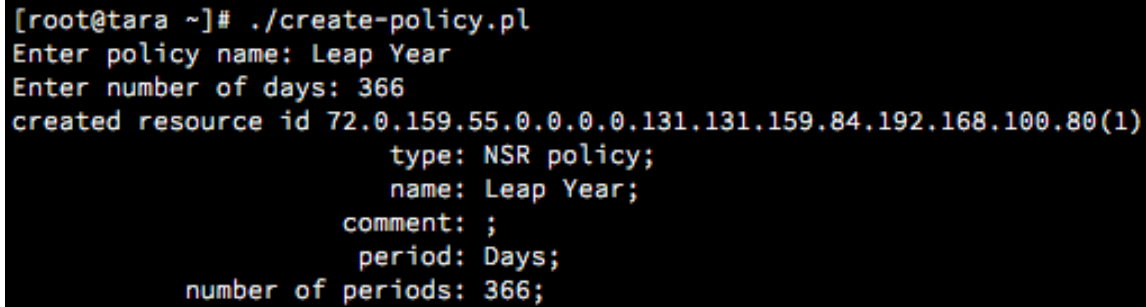
use strict;
print "Enter policy name: ";
my $policyName = <>;
chomp $policyName;

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";
    }
}
```

When executed, this script will produce output such as the following:



```
[root@tara ~]# ./create-policy.pl
Enter policy name: Leap Year
Enter number of days: 366
created resource id 72.0.159.55.0.0.0.0.131.131.159.84.192.168.100.80(1)
        type: NSR policy;
        name: Leap Year;
        comment: ;
        period: Days;
        number of periods: 366;
```

Figure 166: Script to create a new policy on using Perl

15.14.2 Setting up for Scripted Client Creation

Before we configure a script for client creation, we'll create the resources the script will rely on. This will be another good use of creating bulk scripts – rather than creating the resources manually, we'll create a single file with all the setup command required.

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; browse policy: Daily;
retention policy: Daily

create type: NSR group; name: Daily MSSQL; autostart:
Enabled;
start time: "21:55"; schedule: Daily; browse policy: Daily;
retention policy: Daily

create type: NSR group; name: Monthly; autostart: Enabled;
start time: "21:40"; schedule: Monthly; browse policy:
Monthly;
retention policy: Monthly

create type: NSR group; name: Monthly MSSQL; autostart:
Enabled;
start time: "22:00"; schedule: Monthly; browse policy:
Monthly;
retention policy: 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; store index entries: No;
auto media verify: Yes; recycle to other pools: Yes;
recycle from other pools: Yes; retention policy: Daily

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 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 that you can download a zip file with both the Windows and Unix versions of the above script from:

http://nsrd.info/turbocharged/create_resources.zip

To run it, execute:

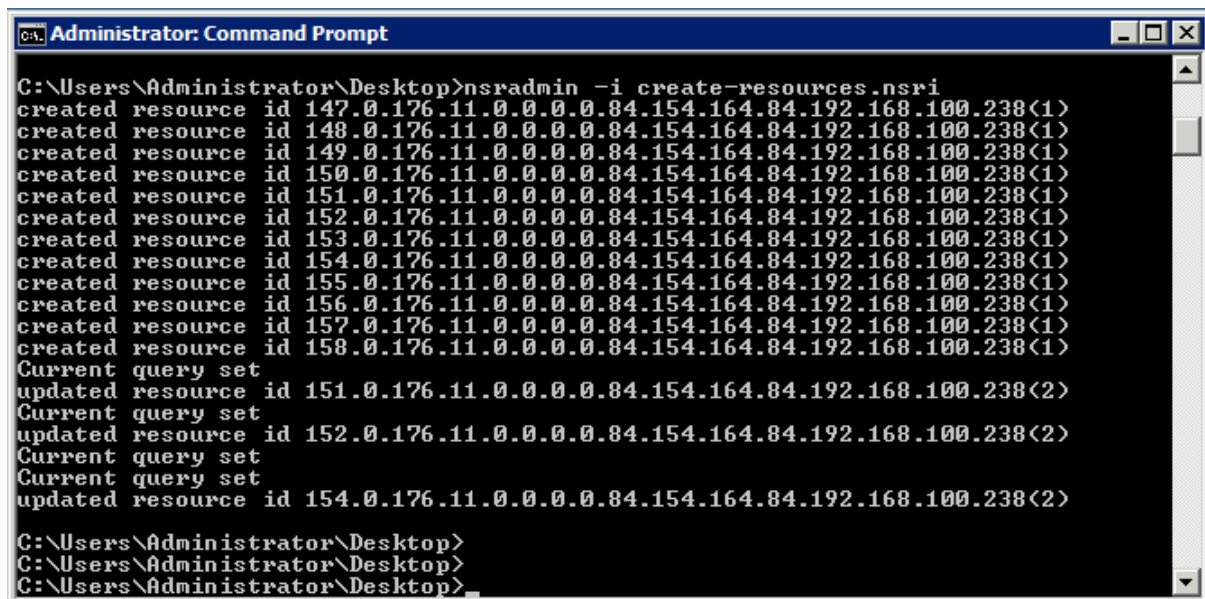
```
# nsradmin -i create-resources.nsri
```

This will execute as follows:

```
[root@tara ~]# nsradmin -i create-resources.nsri
created resource id 146.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 147.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 148.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 149.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 150.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 151.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 152.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 153.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 154.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 155.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 156.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 157.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
Current query set
updated resource id 150.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(2)
Current query set
updated resource id 151.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(2)
Current query set
Current query set
updated resource id 153.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(2)
```

Figure 167: Bulk creation of resources to be used for scripting

The execution will be almost entirely the same on a Windows NetWorker server:



```
Administrator: Command Prompt
C:\Users\Administrator\Desktop>nsradmin -i create-resources.nsri
created resource id 147.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 148.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 149.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 150.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 151.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 152.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 153.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 154.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 155.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 156.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 157.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 158.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
Current query set
updated resource id 151.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<2>
Current query set
updated resource id 152.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<2>
Current query set
Current query set
updated resource id 154.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<2>
C:\Users\Administrator\Desktop>
C:\Users\Administrator\Desktop>
C:\Users\Administrator\Desktop>
```

Figure 168: Bulk creation of resources on Windows

15.14.3 A client creation script

Having tested out a basic script, and created the basic configuration that will be used, we can now create a script that adds clients to our NetWorker environment.

For Linux/Unix, this script might resemble the following:

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

use strict;
my $hostname = "tara";
```

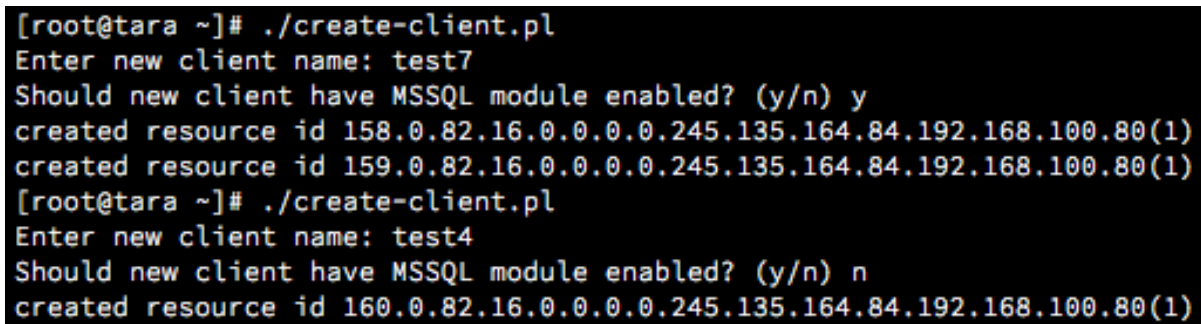
```
print "Enter new client name: ";
my $newClient = <>;
chomp $newClient;

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

if (open(NEWCL,">new-client-$$nsri")) {
    print NEWCL "create type: NSR client; name:
$newClient;\n";
    print NEWCL "group: Daily, Monthly; browse policy:
Monthly;\n";
    print NEWCL "retention policy: Monthly; parallelism:
1\n";
    if ($module eq "y") {
        print NEWCL "create type: NSR client; name:
$newClient;\n";
        print NEWCL "group: Daily MSSQL, Monthly
MSSQL;\n";
        print NEWCL "browse policy: Monthly; retention
policy: Monthly;\n";
        print NEWCL "backup command: nsrsqlsv.exe -s
$server;\n";
        print NEWCL "save set: \"MSSQL:\n";
    }
    close(NEWCL);
    system("nsradmin -s $hostname -i new-client-$$nsri");
    unlink("new-client-$$nsri");
} else {
    die "Could not create file new-client-$$nsri\n";
}
```

Change the *hostname* entry in the line “my \$hostname = ...” to the name of your lab NetWorker server.

Script sessions would resemble the following:



```
[root@tara ~]# ./create-client.pl
Enter new client name: test7
Should new client have MSSQL module enabled? (y/n) y
created resource id 158.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
created resource id 159.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
[root@tara ~]# ./create-client.pl
Enter new client name: test4
Should new client have MSSQL module enabled? (y/n) n
created resource id 160.0.82.16.0.0.0.0.245.135.164.84.192.168.100.80(1)
```

Figure 169: New client script being executed on Linux

In the first execution, two client resources were created – the first for the filesystem backup, and the second for the SQL server backup. For the second session, only a filesystem backup instance was created.

The Windows batch file version of the script will look like the following:

```
@echo off
set server=win01

echo Creating a new client
set /p name="Enter new client name: "
```

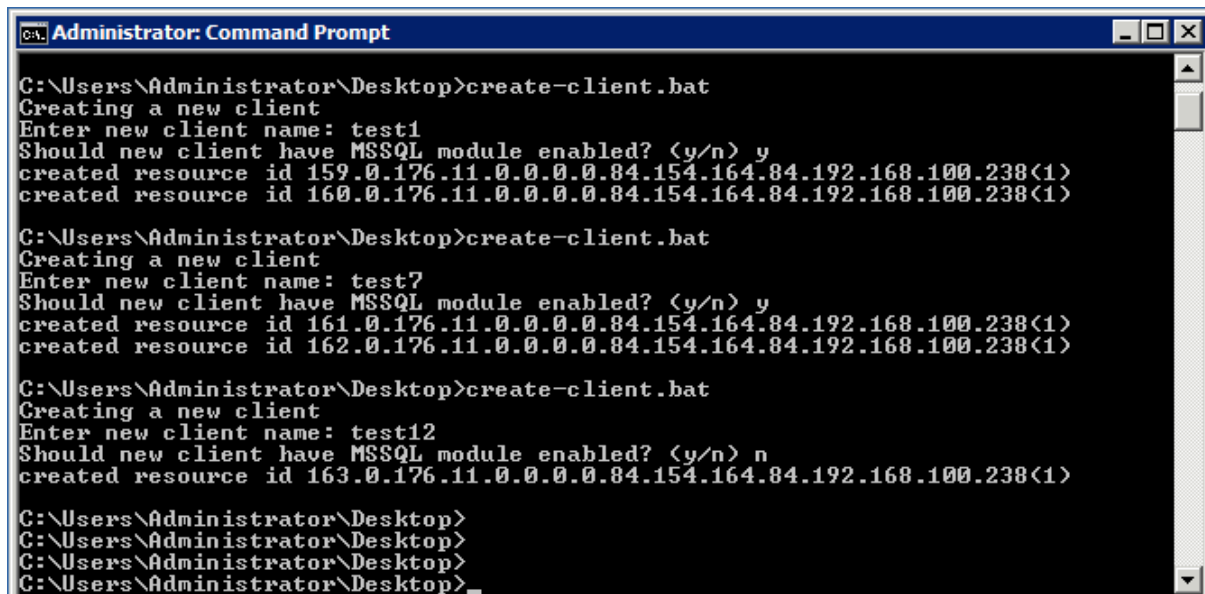
```
set /p module="Should new client have MSSQL module enabled?
(y/n) "

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

nsradmin -s %server% -i command.nsri
del command.nsri
```

When entering the script, be sure to change the server name (“set server=winol”) to the name of your NetWorker lab server. With the script created and saved as “create-client.bat”, sample run session results are as follows:



```
C:\Users\Administrator\Desktop>create-client.bat
Creating a new client
Enter new client name: test1
Should new client have MSSQL module enabled? (y/n) y
created resource id 159.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 160.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>

C:\Users\Administrator\Desktop>create-client.bat
Creating a new client
Enter new client name: test7
Should new client have MSSQL module enabled? (y/n) y
created resource id 161.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>
created resource id 162.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>

C:\Users\Administrator\Desktop>create-client.bat
Creating a new client
Enter new client name: test12
Should new client have MSSQL module enabled? (y/n) n
created resource id 163.0.176.11.0.0.0.0.84.154.164.84.192.168.100.238<1>

C:\Users\Administrator\Desktop>
C:\Users\Administrator\Desktop>
C:\Users\Administrator\Desktop>
C:\Users\Administrator\Desktop>
```

Figure 170: Executing the create-client script on Windows

15.15 Connecting to Client Services

As mentioned in 11.1.2 (Aside – Auto-rendered Log Files), nsradmin can be used to the NetWorker client service as well. The syntax for this was:

```
# nsradmin -s clientName -p nsrexec
```

(Note that you can use the program ‘nsrexec’ or ‘nsrexecd’ here.)

Once connected, you can view the NSRLA resource to view various items of software and operating system configuration for the client. For instance:

```
nsradmin> print type: NSRLA
      type: NSRLA;
      name: faraway;
      nsrmmmd version: ;
      nsrsnmd version: ;
      NW instance info operations: ;
      NW instance info file: ;
      installed products: ;
      auth methods: "0.0.0.0/0,nsrauth/oldauth";
      max auth attempts: 8;
      administrator: "group=Administrators,host=faraway",
                    "group=Administrators,host=localhost",
                    "isroot,host=tara", "isroot,host=tara.pmdg.lab";
      arch: Windows Server 2003;
      kernel arch: INTEL_PENTIUM;
      CPU type: INTEL_PENTIUM;
      machine type: server;
      OS: Windows Server 2003 5.2;
      NetWorker version: 8.1.1.7.Build.333;
      client OS type: Windows NT Server on Intel;
      CPUs: 2;
      MB used: 22702;
      IP address: 192.168.100.7;
```

Figure 171: Using nsradmin to view the NSRLA resource on a client

Pertinent information offered by the client includes:

- Client IP addresses
- Number of CPUs
- Amount of occupied space on the client ("MB used")
- Operating system version
- NetWorker client version
- Authorised administrators
- Allowed authentication types

Another area this can come in handy is fixing *NSR peer information* errors in NetWorker. Whenever a NetWorker host connects to another NetWorker host for the first time, the two hosts exchange certificates – these are typically auto-generated the first time the client service is started on a host (regardless of its function). The certificates are effectively used as a means of authorisation and to try to prevent impersonation scenarios.

However, sometimes the certificates might be regenerated. This can happen when an operating system is rebuilt, and sometimes even if the NetWorker client software is uninstalled then reinstalled.

The peer information that any NetWorker host collects about other hosts it has communicated with can be viewed by running nsradmin against the client services, then running the command:

```
nsradmin> print type: NSR peer information
```

For instance, on the lab NetWorker server used for this section, the peer information reflects only a single Windows client that was backed up:


```
nsradmin> print type: NSR peer information
               type: NSR peer information;
               administrator: root, "user=root,host=tara";
               name: faraway;
               peer hostname: faraway;
               Change certificate: ;
               certificate file to load: ;
```

Figure 172: Viewing the peer certificate information for a client

In circumstances where a NetWorker client reports the peer information for the server is invalid, or where the server reports the peer information for a client is invalid, the information logged will resemble the following:

```
89879 04/11/2014 11:44:06 5 12 10 240222208 58 0 hyperion
nsrexecd GSS critical An authentication request from
tara.pmdg.lab was denied. The 'NSR peer information' provided
did not match the one stored by hyperion. To accept this
request, delete the 'NSR peer information' resource with the
following attributes from hyperion's NSRLA database: name:
tara.pmdg.lab; NW instance ID: 8d92806e-00000004-8ec98419-
54575478-0001b3a0-02efe8cc; peer hostname: tara.pmdg.lab
```

When this occurs, the process for rectifying it is as follows:

1. Use nsradmin to connect to the client services of the host referenced in the error log as having incorrect peer information stored (in the above message, 'hyperion').
2. Delete the peer information for the host referenced as not matching (in this case, 'tara.pmdg.lab').

For the above scenario, this would be executed as follows²⁷:

```
# nsradmin -s hyperion -p nsrexec
nsradmin> delete type: NSR peer information; name:
tara.pmdg.lab
```

15.16 Using regular expressions in nsradmin

Regular expressions in nsradmin are primarily limited to the use of the standard asterisk wild card. However, even that can result in selecting a large number of resources, so be very careful when using wild cards in nsradmin to carefully confirm the resources selected by a regular expression match your intent.

Starting with a basic example – we should have some *testn* clients defined by practicing with the *client-create* script written previously. We can see what clients we've created as follows:

```
nsradmin> option regexp
...
nsradmin> show name;; backup command;; save set;; group:
nsradmin> print type: NSR client; name: test*
```

For example:

²⁷ It would likely be required to execute this *from* the host called 'hyperion'.

```
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: test4;
      group: Daily, Monthly;
      save set: All;
      backup command: ;

      name: test7;
      group: Daily, Monthly;
      save set: All;
      backup command: ;

      name: test7;
      group: Daily MSSQL, Monthly MSSQL;
      save set: "MSSQL:";
      backup command: nsrsqlsv.exe -s test7;
```

Figure 173: Using regular expressions to show clients in nsradmin

In order to use regular expressions, you *must* invoke first turn them on using the “option regexp” setting. Otherwise, NetWorker will interpret what you type literally:

```
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;
```

Figure 174: How NetWorker interprets regular expressions when it isn't expecting them

Equally, the wild card can only be used at the end of an expression, not partway through it. For instance, searching for a client resources of the name `*7` will not work:

```
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: *7
No resources found for query:
                                name: *7;
                                type: NSR client;
```

Figure 175: Limitations with regular expressions in nsradmin

15.17 Offline mode

All the examples presented so far for nsradmin make use of it connecting to an actual server, via commands such as:

```
# nsradmin
```

and

```
C:\> nsradmin -s serverName
```

We can however run it in *offline* mode against a resource configuration database.

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

To look at offline mode, we'll shutdown NetWorker on the lab server. For Linux/Unix servers, this will be by running the commands:

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

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

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

Once the services are stopped, you can run nsradmin against the resource database. This is achieved using the syntax:

```
# nsradmin -d /path/to/nsrddb
```

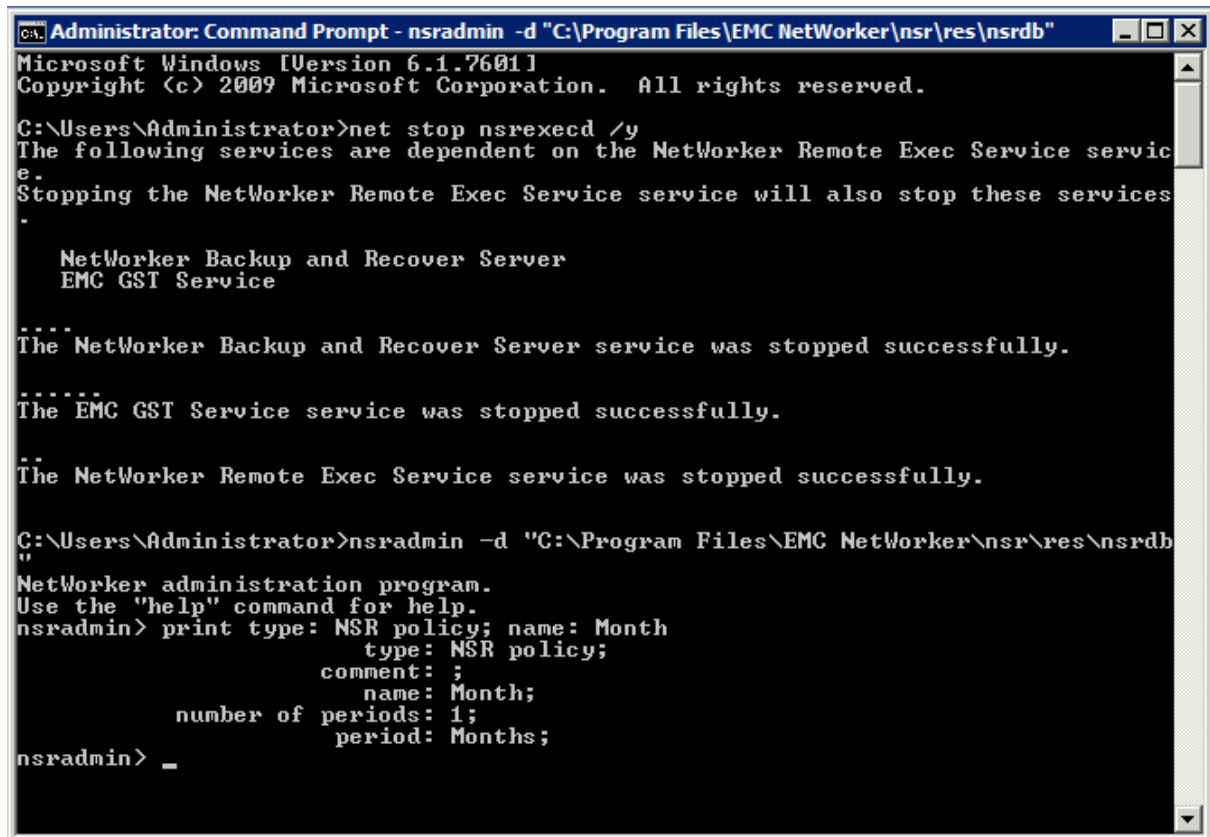
Note that on Unix/Linux systems, nsradmin will often let you get away with using a relative directory path, but in Windows an absolute path is more often required.

An example session on Linux might resemble the following:

```
[root@tara ~]# /etc/init.d/gst stop
Stopping GST: ..
done.
[root@tara ~]# /etc/init.d/networker stop
[root@tara ~]# 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;
```

Figure 176: Running nsradmin in offline mode on Linux/Unix

On Windows, the process will be reasonably similar:



```
C:\Users\Administrator>net stop nsrexecd /y
The following services are dependent on the NetWorker Remote Exec Service service.
Stopping the NetWorker Remote Exec Service service will also stop these services
.
    NetWorker Backup and Recover Server
    EMC GST Service

....
The NetWorker Backup and Recover Server service was stopped successfully.

.....
The EMC GST Service service was stopped successfully.

..
The NetWorker Remote Exec Service service was stopped successfully.

C:\Users\Administrator>nsradmin -d "C:\Program Files\EMC NetWorker\nsr\res\nsrdb"
NetWorker administration program.
Use the "help" command for help.
nsradmin> print type: NSR policy; name: Month
                type: NSR policy;
                comment: ;
                name: Month;
                number of periods: 1;
                period: Months;
nsradmin> _
```

Figure 177: Running nsradmin in offline mode on Windows

That's all for offline mode, for one simple reason: many of the input validation and safety checks performed by the NetWorker server when you work with nsradmin aren't performed when you're operating in offline mode. Therefore, even if you're a power user for nsradmin, you should generally only work in offline mode against a resource database if your support provider gives you specific instructions to do so.

Maintenance

16 Introduction

NetWorker is *mostly* self-maintaining. When you restart the NetWorker server, for instance, it runs a basic check against the media database and the client file indices. Additionally, it routinely runs (daily) media database/index cross-referencing activities essential to making savesets, and by extension, the volumes they may be on, recyclable.

There are two elements in particular for NetWorker maintenance that you should become familiar with, however:

- Specific health-check and processing commands
- Log file control

The activities described in this chapter shouldn't be seen as a complete replacement to whatever health check activities you're already performing, or instructions you may receive from your support provider or the EMC support website. They can, however, be a useful adjunct.

17 Health check commands

Unless otherwise stated, all health check commands are run on a NetWorker server only.

CAUTION – Lab Exercises

All of the exercises described in this chapter should only be attempted on lab servers until you are confident with what you are doing. Depending on the state of your existing NetWorker media database and indices, these commands can result in backups becoming recyclable and purged from a system, or index records becoming unavailable. Use at your own discretion.

17.1 Media Database Check

The *nsrck* utility can be used to perform both media database and index checks. We'll start with the simplest check it can perform – the rebuild of the media database.

The media database check/rebuild is achieved by running:

```
# nsrck -m
```

On older versions of NetWorker, this would provide *no* output to standard out if there were no issues encountered. From NetWorker 8.2 onwards, this reports:

```
[root@centaur ~]# nsrck -m
Succeeded to rebuild Media DB indexes
```

Figure 178: Standard *nsrck* output, starting in NetWorker 8.2

The *nsrck -m* command can perform cumulative repair operations on a media database. If for instance it reports an error, running it another one or two times can result in the error being repaired (or eliminated).

Depending on the size of your media database, this command can return almost instantly, or it can take a few minutes to execute. While it may not appear from the output to be doing much, if we check the *daemon.raw* file (via *nsr_render_log*), the following is reported during an *nsrck -m* operation:

```
0 03/01/15 10:37:51 3 0 0 3759986432 4796 0 centaur nsrmmdbd NSR error 01/03/15 10:37:51 nsrmmdbd: media db manage operation invoked
71193 03/01/15 10:37:51 0 0 0 2353030912 4784 0 centaur nsrd NSR info Media Database Notice: media db manage operation invoked
0 03/01/15 10:37:51 3 0 0 3759986432 4796 0 centaur nsrmmdbd NSR error 01/03/15 10:37:51 nsrmmdbd: media db manage operation completed
71193 03/01/15 10:37:51 0 0 0 2353030912 4784 0 centaur nsrd NSR info Media Database Notice: media db manage operation completed
0 03/01/15 10:37:51 3 0 0 3759986432 4796 0 centaur nsrmmdbd NSR error 01/03/15 10:37:51 nsrmmdbd: media db manage operation invoked
71193 03/01/15 10:37:51 0 0 0 2353030912 4784 0 centaur nsrd NSR info Media Database Notice: media db manage operation invoked
10103 03/01/15 10:37:51 1 8 0 3759986432 4796 0 centaur nsrmmdbd NSR notice media db verifying checksums
10105 03/01/15 10:37:51 1 8 0 3759986432 4796 0 centaur nsrmmdbd NSR notice media db is compressing records
9870 03/01/15 10:37:51 1 8 0 3759986432 4796 0 centaur nsrmmdbd NSR notice media db is consistency checking the database
10111 03/01/15 10:37:51 1 8 0 3759986432 4796 0 centaur nsrmmdbd NSR notice media db is record checking
9777 03/01/15 10:37:51 1 8 0 3759986432 4796 0 centaur nsrmmdbd NSR notice media db is cross checking the save sets
9778 03/01/15 10:37:51 1 8 0 3759986432 4796 0 centaur nsrmmdbd NSR notice media db is cross checking the volumes
0 03/01/15 10:37:51 3 0 0 3759986432 4796 0 centaur nsrmmdbd NSR error 01/03/15 10:37:51 nsrmmdbd: media db manage operation completed
71193 03/01/15 10:37:51 0 0 0 2353030912 4784 0 centaur nsrd NSR info Media Database Notice: media db manage operation completed
```

Figure 179: NetWorker daemon log content from running *nsrck -m*

You should only need to run the *nsrck -m* command before NetWorker upgrades or when directed to by your support provider. Typically you'd be directed to do this when there are errors or issues consistent with problems in either the media database or the client file indices.

The other command you can use to check the status of the media database is the *nsrls* command:

```
# nsrls -m
```

This will output database size and record count details (as well as location) for the media database:

```
[root@orilla ~]# nsrls -m
```

Database id 0: /nsr/mm/mmvolume6

Fid	Size	Count	Name
0	16 KB	2	vol
1	144 KB	194	ss
2	16 KB	7	clients
3	16 KB	2	vol_i0
4	16 KB	1	vol_i1
5	16 KB	1	vol_i2
6	16 KB	1	vol_i3
7	16 KB	0	vol_i4
8	16 KB	194	ss_i0
9	16 KB	194	ss_i1
10	16 KB	0	ss_i2
11	16 KB	194	ss_i3
12	16 KB	7	clients_i0
13	16 KB	7	clients_i1

Figure 180: Using the `nsrls` command against the media database

The usage scenarios for `nsrls -m` are the same as for `nsrck -m`.

17.2 Index Checks

While the previous section covered using `nsrck` and `nsrls` against the media database, both will equally run against the client file indices as well. In fact, both *default* to running against client file indices.

For instance, `nsrck` when run with no arguments will do a level-1 index check, the same as you get when you start the NetWorker server:

```
# nsrck
```

```
[root@orilla ~]# nsrck
nsrck: checking index for 'hyperion'
nsrck: /nsr/index/hyperion contains 2348417 records occupying 459 MB
nsrck: checking index for 'faraway'
nsrck: /nsr/index/faraway contains 1898617 records occupying 230 MB
nsrck: checking index for 'orilla.turbamentis.int'
nsrck: /nsr/index/orilla.turbamentis.int contains 2184695 records occupying 408 MB
nsrck: checking index for 'archon'
nsrck: /nsr/index/archon contains 2294 records occupying 516 KB
nsrck: checking index for 'mondas'
nsrck: /nsr/index/mondas contains 523819 records occupying 80 MB
nsrck: Completed checking 5 client(s)
```

Figure 181: Performing a basic index check

Equally for `nsrls`:

nsrls

```
[root@orilla ~]# nsrls

/nsr/index/archon: 2294 records requiring 516 KB
/nsr/index/archon is currently 100% utilized

/nsr/index/faraway: 1898617 records requiring 230 MB
/nsr/index/faraway is currently 100% utilized

/nsr/index/hyperion: 2348417 records requiring 459 MB
/nsr/index/hyperion is currently 100% utilized

/nsr/index/mondas: 523819 records requiring 80 MB
/nsr/index/mondas is currently 100% utilized

/nsr/index/orilla.turbamentis.int: 2184695 records requiring 408 MB
/nsr/index/orilla.turbamentis.int is currently 100% utilized
```

Figure 182: Performing a basic client file index listing/summary

The index listing via `nsrls` reports the number of records in and current size of each index. Note for all modern NetWorker servers (v7.0 onwards), the index will be reported as “100% utilized”. This is normal²⁸.

The `nsrck` command actually supports a variety of index check levels. These are accessed via `nsrck -Lx` where *x* is a number between 1 and 7. You’ve already seen the output of a level-1 execution: that’s the default when no arguments are supplied.

While all 7 levels of index checking do slightly different things, the levels you’ll most likely invoke *without* direction from your support provider are:

Table 5: Standard `nsrck` check levels

Index Check Level	Description
-L1	Invoked by default if running <code>nsrck</code> without arguments. Basic validation only.
-L3	Cross-references the client file indices against the media database. Where a client file index entry is discovered that has no media database entries (i.e., savesets), the client file index entry is removed.
-L6	Completely rebuild the client file indices. This can clear almost all corruption from an index.
-L7	Rebuild the client file indices by <i>recovering</i> them from backup.

For level based checks, `nsrck` can be invoked against either *all* clients, or a single nominated client, using the syntax:

nsrck -Lx [client]

Where:

- -Lx is the level of check to run

²⁸ Older indices had a different format that allowed for records in an index to be marked as expired, and would be removed from the file on a media/index cross-reference check. This is no longer required as separate index files are stored for each saveset and are automatically removed as required.

- Client (optional) is the specific client to run the check against.

For instance, to perform a level 3 check (index/media database cross reference) against all clients, the syntax would be:

```
# nsrck -L3
```

For example:

```
[root@orilla ~]# nsrck -L3
nsrck: checking index for 'mondas'
nsrck: /nsr/index/mondas contains 523819 records occupying 80 MB
nsrck: checking index for 'orilla.turbamentis.int'
nsrck: /nsr/index/orilla.turbamentis.int contains 2184695 records occupying 408 MB
nsrck: checking index for 'faraway'
nsrck: /nsr/index/faraway contains 1898617 records occupying 230 MB
nsrck: checking index for 'archon'
nsrck: /nsr/index/archon contains 2294 records occupying 516 KB
nsrck: checking index for 'hyperion'
nsrck: /nsr/index/hyperion contains 2348417 records occupying 459 MB
nsrck: Completed checking 5 client(s)
```

Figure 183: Executing an nsrck -L3

To actually rebuild the index for a specific client, inline, you would use the syntax:

```
# nsrck -L6 clientName
```

For example:

```
# nsrck -L6 faraway
```

```
[root@orilla ~]# nsrck -L6 faraway
nsrck: checking index for 'faraway'
nsrck: /nsr/index/faraway contains 1898617 records occupying 230 MB
nsrck: Completed checking 1 client(s)
```

Figure 184: Performing an index rebuild against a client file index

An actual index recovery will look a little different, since nsrck manages the background recovery task for you. For example:

```
# nsrck -L7 faraway
```

```
[root@orilla ~]# nsrck -L7 faraway
nsrck: checking index for 'faraway'
9343:nsrck: The file index for client 'faraway' will be recovered.9433:nsrck: Recovering index savesets of 'faraway' from 'orilla.turbamentis.int'
Recover start time: Sat 03 Jan 2015 11:04:46 AEDT
Requesting 1 recover session(s) from server.
91651:recover: Successfully established AFTD DFA session for recovering save-set ID '3685139683'.
Recover completion time: Sat 03 Jan 2015 11:07:55 AEDT
9346:nsrck: completed recovery of index for client 'faraway'
nsrck: /nsr/index/faraway contains 1898617 records occupying 230 MB
nsrck: Completed checking 1 client(s)
```

Figure 185: Performing an index recovery

Note that like any normal recovery, NetWorker will recover all components necessary – for index backups, this will be the most recent full and the most recent level-9 backup performed of the index.

Examples of when you'd run these sorts of nsrck commands are:

Table 6: Scenarios for running nsrck index rebuilds

Index Check Level	When you'd run...
-L1	Automatically run every time you start NetWorker.
-L3	When advised by support or in scenarios where NetWorker is reporting index problems when attempting a recovery.
-L6	When advised by support or in scenarios where NetWorker is reporting index problems when attempting a recovery and the -L3 option has not solved the problem.
-L7	When advised by support, or in scenarios where you need to merge in older indices, or when you're performing a disaster recovery of either the NetWorker server or a specific client's indices.

Note in the -L7 description above the reference to merging in old indices. The syntax for this type of index recovery is:

```
# nsrck -t date -L7 clientName
```

Where *date* is a valid NetWorker date. This will retrieve older index backups from the nominated time and merge them with the current index, effectively allowing you to retrieve browseable index details for older, non-browseable backups, or backups whose browse details have been mistakenly purged from the client file index.

17.3 Index Management

There is another utility you should be aware of, but careful as to when or if you use it, and that's *nsrim -X*. In normal operations it should be rare indeed for you to have to run it.

The *nsrim* utility is used to perform NetWorker index management – it's the utility used by NetWorker to expire old backups then invoke *nsrck* as required to clean up the client file indices, before (if possible) marking volumes as recyclable.

You should **never, ever** run *nsrim* against a server where there are backup, clone or staging activities running on a system unless specifically advised by your support provider.

The complete output of *nsrim -X* is quite comprehensive and is shown below in trimmed form from a lab server:

```
[root@orilla ~]# nsrim -X
88411:nsrim: Checking for invalid volumes
86069:nsrim: Processing 7 clients
faraway:C:\, 5 browsable cycle(s)
1820371 browsable files of 1820371 total, 61 GB recoverable
of 61 GB total

faraway:D:\, 5 browsable cycle(s)
21974 browsable files of 21974 total, 9753 MB recoverable of
9753 MB total
```

...the above output is repeated in style for each client:saveset combination...

```
86067:nsrim: Crosschecking indexes for 4 clients.
Cross checking client(s):
    faraway
    hyperion
    mondas
    orilla.turbamentis.int
nsrck: checking index for 'faraway'
```

```
nsrck: /nsr/index/faraway contains 1898617 records occupying
230 MB
nsrck: checking index for 'hyperion'
nsrck: /nsr/index/hyperion contains 2348417 records occupying
459 MB
nsrck: checking index for 'mondas'
nsrck: /nsr/index/mondas contains 523819 records occupying 80
MB
nsrck: checking index for 'orilla.turbamentis.int'
nsrck: /nsr/index/orilla.turbamentis.int contains 2184695
records occupying 408 MB
nsrck: Completed checking 4 client(s)
86068:nsrim: Managing 5 volumes.
DAS.01: 530 GB used, 42 save sets, appendable, 28
browsable save sets, 14 recoverable save sets
DAS.02: 0 KB used, 0 save sets, appendable
iSCSI.01: 1992 GB used, 271 save sets, appendable, 182
browsable save sets, 88 recoverable save sets, 1 recyclable
save sets
iSCSI_SDFS.01: 53 GB used, 146 save sets, appendable, 101
browsable save sets, 45 recoverable save sets
iSCSI_SDFS.02: 20 GB used, 18 save sets, appendable, 16
browsable save sets, 2 recoverable save sets
86073:nsrim: Compressing media database.
```

Typically you should only run *nsrim -X* when advised to by your support provider, but one example where you *may* have to run it is when you're using tapes and you need to trigger a recycling check to determine if there is any media eligible for recycling. NetWorker upgrade instructions will also require this utility to be executed at a suitable point in the process, too.

18 Client Connectivity Checking

Introduced in the NetWorker 8.x series has been a new function in the *nsradmin* utility for checking configured clients. This can be an absolute boon to administrators trying to diagnose connectivity issues within their environment.

This client check will, for each client targeted, gather the following information:

- Client name
- Client ID
- Client FQDN
- IP Address(es)
- Reverse Lookup Results
- Client port connectivity check results

The syntax for this new function is:

```
# nsradmin -C query
```

Where *query* is any valid NetWorker query that identifies clients. (The goal over time is to allow other types of checks to be performed for a variety of NetWorker resources, but at the moment the first function available – arguably the most important one – is for client checking.)

If you wanted to check *all* clients in your environment, the syntax might resemble the following:

```
# nsradmin -C "type: NSR client"
```

However, if you've got a lot of clients, you might want to restrict this a little – such as by *group* the client belongs to. So to query all the clients that belong to a group called *Servers*, for instance, the command might resemble the following:

```
[root@orilla ~]# nsradmin -C "type: NSR client; group: Servers"

Validate "NSR client" resources

Synopsis: For each NSR client resource in orilla.turbamentis.int's NSR
database, verify their 'name', 'aliases', 'storage nodes' and 'server
network interface' attributes have properly configured DNS entries then
attempt to connect to each address on port 7938.

Client 1 of 3
Name: faraway
Client ID: 76235b01-00000004-54584ba8-54597338-002cb3a0-02efe8cc

Canonical hostname:      faraway.turbamentis.int
IP Address:              192.168.100.7 (0.000 sec)
Host Name (reverse lookup): faraway.turbamentis.int (0.000 sec)
Ping (port 7938):        Success (0.000 sec)

Alias: faraway.turbamentis.int

Canonical hostname:      faraway.turbamentis.int
IP Address:              192.168.100.7 (0.000 sec)
Host Name (reverse lookup): faraway.turbamentis.int (0.000
sec)
Ping (port 7938):        Success (0.000 sec)

Client 2 of 3
Name: mondas
Client ID: 147f6a46-00000004-5457fce2-5457fcel-0016b3a0-02efe8cc

Canonical hostname:      mondas.turbamentis.int
IP Address:              192.168.100.99 (0.000 sec)
Host Name (reverse lookup): mondas.turbamentis.int (0.000 sec)
Ping (port 7938):        Success (0.000 sec)

Alias: mondas.turbamentis.int

Canonical hostname:      mondas.turbamentis.int
IP Address:              192.168.100.99 (0.000 sec)
Host Name (reverse lookup): mondas.turbamentis.int (0.000
sec)
Ping (port 7938):        Success (0.000 sec)

Client 3 of 3
Name: orilla.turbamentis.int
Client ID: fd071e91-00000004-5457547a-54575479-0001b3a0-02efe8cc

Canonical hostname:      orilla.turbamentis.int
IP Address:              192.168.100.4 (0.000 sec)
Host Name (reverse lookup): orilla.turbamentis.int (0.000 sec)
Ping (port 7938):        Success (0.000 sec)

Alias: orilla

Canonical hostname:      orilla.turbamentis.int
IP Address:              192.168.100.4 (0.000 sec)
Host Name (reverse lookup): orilla.turbamentis.int (0.000
sec)
Ping (port 7938):        Success (0.000 sec)

Summary:

NSR client resources checked:      3
Names checked:                    3
  Forward lookup errors:          0
  Reverse lookup errors:          0
  Ping errors:                    0
Aliases checked:                  3
```

```
Forward lookup errors:      0
Reverse lookup errors:     0
Ping errors:               0

Server Network Interfaces checked: 0
Forward lookup errors:     0
Reverse lookup errors:     0
Ping errors:               0

Storage Nodes checked:     0
Forward lookup errors:     0
Reverse lookup errors:     0
Ping errors:               0

Total errors:              0
```

Even if you ignore most of the other functionality of nsradmin, this is an option you should *definitely* familiarise yourself with.

19 Using dbgcommand

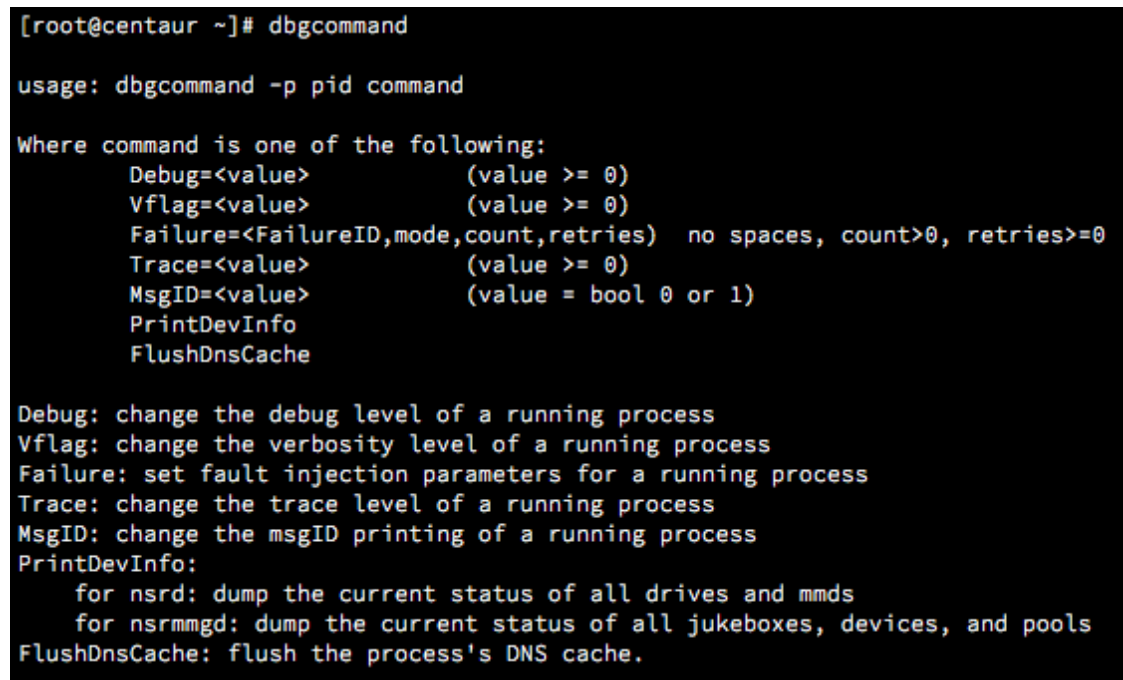
The *dbgcommand* was introduced in the NetWorker 7.x tree (formally introduced into the NetWorker distribution in v7.4, previously supplied by support when required), and is used to put specific NetWorker services into debug mode without needing to restart NetWorker services.

Additionally, the *dbgcommand* has several nifty options that can make problem diagnosis easier to perform.

CAUTION - Use in lab first

The *dbgcommand* while extremely useful can also cause problems if used incorrectly. You are strongly advised to ensure you thoroughly familiarise yourself with it in a lab environment before using it in a production environment, and for the most part limit its use in a production environment to specific support situations.

Invoked without any arguments, *dbgcommand* will produce output such as the following:



```
[root@centaur ~]# dbgcommand

usage: dbgcommand -p pid command

Where command is one of the following:
  Debug=<value>           (value >= 0)
  Vflag=<value>           (value >= 0)
  Failure=<FailureID,mode,count,retries> no spaces, count>0, retries>=0
  Trace=<value>           (value >= 0)
  MsgID=<value>           (value = bool 0 or 1)
  PrintDevInfo
  FlushDnsCache

Debug: change the debug level of a running process
Vflag: change the verbosity level of a running process
Failure: set fault injection parameters for a running process
Trace: change the trace level of a running process
MsgID: change the msgID printing of a running process
PrintDevInfo:
  for nsrd: dump the current status of all drives and mmds
  for nsrmmgd: dump the current status of all jukeboxes, devices, and pools
FlushDnsCache: flush the process's DNS cache.
```

Figure 186: Default output of *dbgcommand*

19.1 Correlating devices to running daemons

Particularly in a tape-based environment, it's useful to be able to cross reference a running *nsrmmd* process to the device it is managing. For example, if we look at a *ps* output for *nsrmmd* processes, it will resemble the following:

```
[root@centaur ~]# ps -eaf | grep nsrmmd
root      2349   2297   0 Mar27 ?        00:00:27 /usr/sbin/nsrmmdbd
root      17329  2574   0 Mar29 ?        00:00:14 /usr/sbin/nsrmmd -b 2 -N 3409981
45 -n 8 -s centaur -t centaur
root      23086  2574   0 10:30 ?        00:00:00 /usr/sbin/nsrmmd -b 2 -N 3409981
45 -n 5 -s centaur -t centaur
root      23243  2574   0 10:30 ?        00:00:00 /usr/sbin/nsrmmd -b 2 -N 3409981
45 -n 7 -s centaur -t centaur
root      23282  2574   0 10:30 ?        00:00:00 /usr/sbin/nsrmmd -b 2 -N 3409981
45 -n 20 -s centaur -t centaur
root      23405  2574   0 10:31 ?        00:00:00 /usr/sbin/nsrmmd -b 2 -N 3409981
45 -n 21 -s centaur -t centaur
root      23581  2574   0 10:31 ?        00:00:00 /usr/sbin/nsrmmd -b 2 -N 3409981
45 -n 22 -s centaur -t centaur
```

Figure 187: *ps* output for *nsrmmd* processes

The *dbgcommand* makes the cross referencing considerably easier. This is done by executing:

```
# dbgcommand -p nsrdPID PrintDevInfo
```

Where *nsrdPID* is the process ID for the *nsrd* executable (or *nsrd.exe* on a Windows NetWorker server²⁹). This might resemble the following:

```
[root@centaur ~]# ps -eaf | grep nsrd
root      2297     1   0 Mar27 ?        00:03:12 /usr/sbin/nsrd
root      2410   2297   0 Mar27 ?        00:00:09 /usr/sbin/nsrdispd
root      2963   2410   0 Mar27 ?        00:00:32 /usr/sbin/nsrdisp_nwbgs
root      24413 21150   0 11:12 pts/0    00:00:00 grep nsrd
[root@centaur ~]# dbgcommand -p 2297 PrintDevInfo
```

Figure 188: Generating device information using *dbgcommand*

You'll note if you run this command that it doesn't produce any terminal output – this is because the information generated is written to the *daemon.raw* file.

The output generated by the *dbgcommand* is too long to include in this document, but for each device a reasonably substantial amount of information is produced. If the device is active or permanent *nsrmmds* are assigned, you'll see included in the output information such as the following:

```
device squeezebox_Squeeze01 {
    d_mmd_list :
        count = 1
        mmd_number = 340998152
    d_iface = DEV_IF_DD_IP
    d_dedicated_snode = <NULL>
    d_pid = 0
    d_nsrmmd_number = 0
    d_mode = M_RW
    rm_soft = RDS_ENABLED
```

²⁹ This can be determined through a variety of means in Windows – in the simplest form, just adding the PID column to the Activity Monitor process display.


```
d_read_only = <FALSE>
d_jbdev = <FALSE>
d_device = squeezebox_Squeeze01
d_family = disk
...
```

Included in the output is the list of nsrmmmd processes based on the internal NetWorker reference number that are managing or working on the device. In this case, that's 340998152, which has been underlined in the output above. Looking further through the daemon.raw output you'll see a section for this specific mmd that resembles the following:

```
mmd #340998152 {
  mm_flags = 0
  mm_number = 340998152
  mm_active = 0
  mm_operation = RM_INIT
  mm_idle_time = Mon Apr  6 10:45:56 GMT+1000 2015
  mm_mode = MM_MODE_INIT
  mm_pid = 17329
  mm_volume =
  mm_device = <NULL>
  mm_pool =
  mm_machname = centaur
  mm_control = MM_READY
  mm_auth {
    mm_auth_session = 0
    mm_auth_clone_partner = 0
    mm_auth_clone_mount_id = 0
    mm_auth_mode = MM_MODE_INIT
    mm_auth_timeout = 0
    mm_auth_setup_timeout = 0
  }
  mm_minmode = _DONE
  mm_save_lockout = 0
  mm_ndmp = <FALSE>
  mm_hard_limit = 0
  mm_is_32bit = <FALSE>
  mm_agent_pid = 0
  mm_b2d_dynamic = <FALSE>
  mm_b2d_device = <NULL>
  reservations {
  }
}
```

Included in *that* output is the process ID for the nsrmmmd process in question – in this case, 17329, and if we do a process listing searching for that specific process ID, we'll find the nsrmmmd in question:

```
[root@centaur ~]# ps -eaf | grep 17329
root      17329  2574  0 Mar29 ?           00:00:14
/usr/sbin/nsrmmmd -b 2 -N 340998145 -n 8 -s centaur -t centaur
```

In older environments that had tape libraries, this cross-referencing was particularly handy to NetWorker administrators who encountered a significant error on a single tape and wanted to try to kill that process and release a tape drive without being forced to restart the NetWorker services. While the results might sometimes be variable, it at least gave an administrator a 'fighting chance' to recover from a significant low-level SCSI error without having to completely restart the NetWorker services.

19.2 Flushing NetWorker's Internal DNS Cache

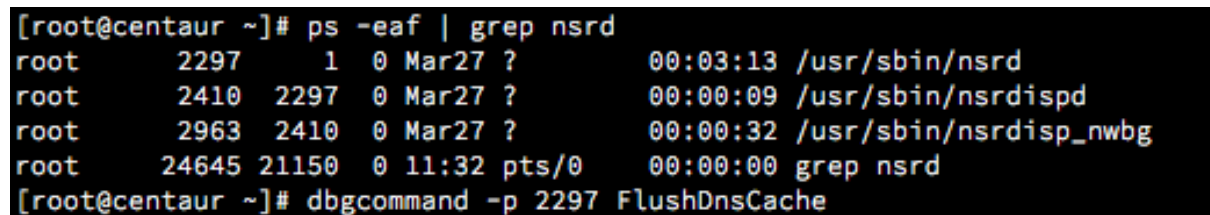
In order to speed up various name resolution activities, NetWorker typically maintains an internal DNS cache of hostnames it has validated. Occasionally this DNS cache may become out of sync with your main DNS – though this typically won't happen unless there are DNS errors or hosts/DNS conflicts.

In this scenario, rather than restarting the NetWorker services, you can use *dbgcommand* to flush that internal cache using the command:

```
# dbgcommand -p nsrdPID FlushDnsCache
```

Where *nsrdPID* is the process ID for the NetWorker core service, *nsrd/nsrd.exe*.

As per the *PrintDevInfo* command, this will produce no discernible output to the terminal it is run on:



```
[root@centaur ~]# ps -eaf | grep nsrd
root      2297      1  0 Mar27 ?        00:03:13 /usr/sbin/nsrd
root      2410    2297  0 Mar27 ?        00:00:09 /usr/sbin/nsrdispd
root      2963    2410  0 Mar27 ?        00:00:32 /usr/sbin/nsrdisp_nwbg
root      24645  21150  0 11:32 pts/0    00:00:00 grep nsrd
[root@centaur ~]# dbgcommand -p 2297 FlushDnsCache
```

Figure 189: Using the *FlushDnsCache* option for *dbgcommand*

However, there will be a notice generated in the *daemon.raw* file that resembles the following:

```
Changing FlushDnsCache of process (id 2297) from 0 to 1
```

(You may note that like the *daemon.raw* content generated for the *PrintDevInfo* command, this content is not written in a way that *requires* rendering.)

19.3 Turning on Debug Mode

Occasionally EMC support may request you use the *dbgcommand* utility to actually turn on debug mode for a specific NetWorker process. While this will *usually* be the *nsrd/nsrd.exe* process, it can conceivably be just about any running NetWorker process in your environment, depending on the nature of the problem and what diagnostic work has already been run.

The syntax for this option is:

```
# dbgcommand -p pid Debug=X
```

Where *pid* is the process ID to be changed and *X* is a number between 0 and 99. As you might imagine, a value of 0 turns debug mode off, and you should always be certain to turn debug mode off as soon as you've gathered the information you need, otherwise you may encounter considerable log growth. While there are a variety of different debug numbers you may be called on to use, the most common two are usually 3 and 9.

For example, putting the NetWorker core service, *nsrd* into debug mode level 9, might be done as follows:

```
[root@centaur ~]# ps -eaf | grep nsrd
root      2297      1   0 Mar27 ?          00:03:13 /usr/sbin/nsrd
root      2410    2297   0 Mar27 ?          00:00:09 /usr/sbin/nsrdispd
root      2963    2410   0 Mar27 ?          00:00:32 /usr/sbin/nsrdisp_nwbg
root      24670  21150   0 11:40 pts/0      00:00:00 grep nsrd
[root@centaur ~]# dbgcommand -p 2297 Debug=9
[root@centaur ~]#
```

Figure 190: Turning on debug mode level 9

Whenever a daemon has its debug level changed, this is reflected in the daemon.raw file:

Changing Debug level of process (id 2297) from 0 to 9

Immediately after debug mode has been turned on, the NetWorker services will start generating potentially a lot more messages into the target log file(s). For example, running an *nsr_render_log* immediately after turning on debug mode level 9 might reveal logged output such as the following:

```
Unable to render the following message: Changing Debug level of process (id 2297
) from 0 to 9

0 06/04/15 11:40:17  1 5 0 835585792 2297 0 centaur nsrd NSR notice 04/06/15 11:
40:17.962598 Entering check_requests_queue().
0 06/04/15 11:40:17  1 5 0 835585792 2297 0 centaur nsrd NSR notice 04/06/15 11:
40:17.963061 Entering check_broker_out_queue().
0 06/04/15 11:40:17  1 5 0 835585792 2297 0 centaur nsrd NSR notice 04/06/15 11:
40:17.963087 Exiting check_broker_out_queue().
0 06/04/15 11:40:19  1 5 0 835585792 2297 0 centaur nsrd NSR notice 04/06/15 11:
40:19.005952 Entering check_requests_queue().
```

Figure 191: Debug log information

For the most part, debug information is entirely intended to be read by an EMC support engineer, but can in certain circumstances help to identify scenarios where processes are looping on name resolution attempts, etc.

When ready to exit debug mode for logging, the command might resemble the following:

```
[root@centaur ~]# ps -eaf | grep nsrd
root      2297      1   0 Mar27 ?          00:03:13 /usr/sbin/nsrd
root      2410    2297   0 Mar27 ?          00:00:09 /usr/sbin/nsrdispd
root      2963    2410   0 Mar27 ?          00:00:32 /usr/sbin/nsrdisp_nwbg
root      24701  21150   0 11:45 pts/0      00:00:00 grep nsrd
[root@centaur ~]# dbgcommand -p 2297 Debug=0
[root@centaur ~]#
```

Figure 192: Turning debug mode off

20 Log Maintenance

NetWorker can, at times, produce a lot of logging information, and the amount of information produced will increase as the number of backups you perform increases.

The bulk of logs produced by NetWorker will, by and large, reside on the NetWorker server within the logs directory:

- Unix/Linux default path: */nsr/logs*

- Windows default path: C:\Program Files\EMC NetWorker\nsr\logs

However, other common location for log files in a NetWorker environment are:

- The log directory (per the above) on any client
- The 'applogs' directory on any client running a NetWorker module
- The 'logs' directory within the NetWorker Management Console directory

NetWorker will routinely cycle the **daemon.raw** logs, starting a new log and renaming the old log to indicate the last dated event covered, but other logs are not always processed. It's very much worthwhile keeping an eye on the sizes of these logs, though it's arguably the case that no system should be so tight on space that you have to do this on a daily, weekly or perhaps even monthly basis.

Within the NetWorker server logs directory, log files that *aren't* rotated/cycled are the messages file and the rap.log file. On most Unix systems now, the *messages* file will be blank, as NetWorker will redirect messages to the system messages file. (However, this file on older servers could grow to huge sizes.)

The most important thing to keep in mind with most NetWorker logs is you should preserve them, even if you compress them after they've been rotated/cycled. This allows easier retrieval at a later time from backup any logs that correspond to backups you may want to recover from.

Logs should also not be rotated/cycled while the NetWorker server is running. For instance, if we use the *fuser* command on a Linux NetWorker server to check for process IDs accessing the /nsr/logs/rap.log file, we see it's in use by the NetWorker server:

```
[root@orilla ~]# fuser /nsr/logs/rap.log
/nsr/logs/rap.log: 3654
[root@orilla ~]# ps -eaf | grep 3654
root      3654      1  0   2014 ?        00:10:36 /usr/sbin/nsrd
root      3713    3654  0   2014 ?        00:00:21 /usr/sbin/nsrmmdbd
root      3912    3654  0   2014 ?        00:00:01 /usr/sbin/nsrindexd
root      4082    3654  0   2014 ?        00:00:03 /usr/sbin/nsrdispd
root      4219    3654  0   2014 ?        00:01:34 /usr/sbin/nsrjobd
root      4434    3654  0   2014 ?        00:01:26 /usr/sbin/nsrvmsd
root      63574  63209  0  11:34 pts/2    00:00:00 grep 3654
```

Figure 193: Log file in use

Were you to say, delete a rap.log file for being too large, or rename to it something like rap_001_old.log *while the server was running*, NetWorker would likely lose connection to the rap.log file while still thinking it had an open file handle, and information that *would* have been logged to the file would *not* be. (This isn't a NetWorker limitation as such but a user error based on how most operating systems deal with open file handles.)

Backup Control

21 Introduction

This backup control section is the main area where we'll be making use of the NetWorker Management Console. Not everything has to be done at the command line to exert expert control over NetWorker, and controlling the finer points of how backups are executed can be easier and more efficient using NMC.

22 Pre and Post Processing Commands

22.1 Advantages of pre and post processing

Sometimes it's not possible to achieve your backup as a simple filesystem or database backup. Consider the following scenarios:

1. A network switch can generate a loadable dump of its configuration but obviously can't have the NetWorker client software on it.
2. A DMZ host can generate a recoverable dump of its key configuration details but security requirements do not allow backup software to be installed on the host.
3. A full end-of-year backup for a database server (that is normally backed up hot) must be performed with the database shutdown.

In each of these scenarios, pre and post processing can conceivably be used to achieve the backup result:

1. If the network switch allows ssh, a NetWorker client backup might start by issuing an ssh command to the switch to dump its configuration. This output would then be saved to the NetWorker client as a text file and picked up as part of the standard filesystem backup.
2. Similarly to the network switch, the security team may allow the DMZ host to have a single nominated host from within the corporate environment connect to it. This host could have the NetWorker client software installed. When the backup is executed on the client, the client reaches out to the DMZ host, retrieves a previously executed backup, saves that backup to the local filesystem and picks it up as part of the NetWorker backup.
3. The end-of-year database backup might first shutdown the database, then perform a standard filesystem backup, and then restart the database at the conclusion of the backup.

22.2 savenpc

For a very long time, NetWorker's support of pre and post processing was via a mechanism referred to as *savenpc*.

Pre and post processing was achieved by changing the backup command for a client to *savenpc*, which would subsequently generate a *groupname.res* file on the NetWorker client the next time its backup was run. This *groupname.res* file would have the following content:

```
precmd: "echo hello";
pstcmd: "echo bye", "/bin/sleep 5";
timeout: "12:00:00";
abort precmd with group: No;
```

By now the layout of this file should look fairly familiar - it's basically a NetWorker resource file, but in this case just a standalone single-purpose resource. If you didn't want to wait for the file to be created, you could create it yourself using the format above, preserving the formatting exactly.

Once the file was created, you'd edit the file and insert your own pre commands or post commands as required.

While *savenpc* worked, it was somewhat fiddly at times, and it also only worked for filesystem backups. If clients using a NetWorker module required pre and post commands, those pre and post commands would need to be handled by the module.

Because *savenpc* isn't required any more for pre and post command execution from NetWorker 8.2 onwards, we won't be dealing with this in the manual.

22.3 The new order

Starting in NetWorker 8.2, the NetWorker client definition now allows for the direct specification of pre and post commands within the client definition:

The screenshot shows the 'Create Client' dialog box with the following sections:

- Access:** Remote user: [text field], Password: [text field]
- Backup:** Backup command: [text field], Pre command: [text field], Post command: [text field], Save operations: [text field], NAS device: [checkbox], NDMP: [checkbox], NDMP array name: [text field], Application information: [text area]
- Deduplication:** Data Domain backup: [checkbox], Data Domain interface: [dropdown menu (IP)], Avamar deduplication backup: [checkbox], Avamar deduplication node: [dropdown menu]
- Probe:** Probe resource name: [dropdown menu]
- Proxy Backup:** None (selected), SCSi, VMWare, Proxy host: [text field]

Buttons at the bottom: OK, Reset, Cancel.

Figure 194: Specifying pre and/or post commands in NetWorker 8.2+

The following rules apply to pre and post commands created for a client:

1. The name of the command must start with 'nsr' or 'save'.
2. The command must be specified as a plain base filename rather than a path.
3. The command must exist in a default directory path accessible by the NetWorker client. (For least fuss, this should be in the actual NetWorker binary directory.)

For instance, consider scenario 2 in the introduction – that being a DMZ host where the NetWorker client software can't be installed, but ssh to the host is permitted.

In this scenario, we might configure a client called 'mondas' to automatically retrieve the dump file generated on the DMZ host as its pre command:

Figure 195: Configuring a client pre command

Assuming ssh keys have been exchanged between mondas and the DMZ host (which we'll refer to as cerberus in this example), the 'nsr_retrieve_cerberus.sh' file might look like the following:

```
#!/bin/bash

/usr/bin/scp root@cerberus:/.backups/latest.zip
/cerberus/dmz_backup.zip
```

The *nsr_retrieve_cerberus.sh* file has been placed in the /usr/sbin directory on the client mondas.

After a backup has completed for mondas, assuming ssh has been configured correctly, the file should have been successfully transferred *and* backed up. For instance, the following shows a directory listing in the /cerberus directory on mondas first before, then after the backup:

```
[root@mondas cerberus]# pwd && ls -al
/cerberus
total 12
drwxr-xr-x  2 root root 4096 Jan  2 10:00 .
drwxr-xr-x 27 root root 4096 Jan  2 09:58 ..
[root@mondas cerberus]# pwd && ls -al
/cerberus
total 124512
drwxr-xr-x  2 root root      4096 Jan  2 10:02 .
drwxr-xr-x 27 root root      4096 Jan  2 09:58 ..
-rw-r--r--  1 root root 127355629 Jan  2 10:02 dmz_backup.zip
```

Figure 196: Results of pre command

We can further verified this was copied *before* the backup by executing a recovery for it:

```
[root@mondas cerberus]# recover -s orilla
Current working directory is /cerberus/
recover> ls -al
total 124500
-rw-r--r-- root      127355629 Jan 02 10:02 dmz_backup.zip
recover> add dmz_backup.zip
1 file(s) marked for recovery
recover> force
  will overwrite any existing files.
recover> recover
Recovering 1 file into its original location
Volumes needed (all on-line):
      iSCSI_SDFS.01 at iSCSI_SDFS_1TB_01
Total estimated disk space needed for recover is 124 MB.
Requesting 1 file(s), this may take a while...
Recover start time: Fri 02 Jan 2015 10:08:02 AEDT
Requesting 1 recover session(s) from server.
./dmz_backup.zip
39571:recover: ./dmz_backup.zip: file exists, overwriting
Received 1 file(s) from NSR server 'orilla'
Recover completion time: Fri 02 Jan 2015 10:09:02 AEDT
```

Figure 197: Recovering a file transferred as part of a pre command

Obviously it's not desirable to perform a recovery every time a pre command backup is executed in NetWorker in order to confirm files or data generated by that pre-command were backed up. NetWorker will therefore rely on the exit status of the pre command to report whether or not the pre command was executed successfully, and this will be reported in the savegroup completion output:

```
NetWorker savegroup: (notice) Servers-SDFS completed, Total 1 client(s), 1 Succeeded with warning(s). See group completion details for more information.
```

```
Succeeded with warning(s): mondas
```

```
Start time: Fri Jan 2 10:02:16 2015  
End time: Fri Jan 2 10:03:59 2015
```

```
--- Successful Save Sets ---
```

```
* mondas:precmd Exited with exit code: 0, completion severity: INFORMATION(10), completion status: succeeded(0)  
* mondas:All savefs mondas: succeeded.
```

Figure 198: Savegroup completion report showing successful pre command execution

When using the pre command option in NetWorker, NetWorker will *not* run the backup command if the pre command returns unsuccessfully. For instance, if we change the shell script used to scp files across from cerberus to have an incorrect filename, the backup fails almost immediately and gives the following error message as part of the savegroup completion:

```
NetWorker savegroup: (alert) Servers-SDFS completed, Total 1 client(s), 1 Failed. See group completion details for more information.
```

```
Failed: mondas
```

```
Start time: Fri Jan 2 10:21:05 2015  
End time: Fri Jan 2 10:21:13 2015
```

```
--- Unsuccessful Save Sets ---
```

```
* mondas:precmd scp: /.backups/filenotfound.zip: No such file or directory  
* mondas:/ scp: /.backups/filenotfound.zip: No such file or directory  
* mondas:/d/01 scp: /.backups/filenotfound.zip: No such file or directory  
* mondas:/boot scp: /.backups/filenotfound.zip: No such file or directory  
* mondas:/d/backup scp: /.backups/filenotfound.zip: No such file or directory
```

Figure 199: Pre command failure

By the way – if you’re worried from the above output that NetWorker executes the pre command for *each* saveset – don’t! NetWorker only executes the pre command once.

Equally, NetWorker will fail the backup if a post command is specified and cannot be found or executed successfully:

```
NetWorker savegroup: (alert) Servers-SDFS completed, Total 1 client(s), 1 Failed. See group completion details for more information.
```

```
Failed: mondas
```

```
Start time: Fri Jan 2 10:27:20 2015  
End time: Fri Jan 2 10:30:51 2015
```

```
--- Unsuccessful Save Sets ---
```

```
* mondas:postcmd 97277:nsrpost: Launching the command 'nsr_filenotfound.sh'.  
* mondas:postcmd /bin/sh: nsr_filenotfound.sh: command not found
```

Figure 200: Post command failure

23 NetWorker Directives

23.1 Overview

A directive, in NetWorker, is a means of exerting granular control over what gets backed up, and *how* it gets backed up, at the filesystem level. Directives typically do not apply for module backups or image level virtual machine backups (via VBA, VADP or VCB).

We'll start by looking at one of the default directives defined within NetWorker, *Unix Standard Directives*. This has the following content:

```
<< / >>
    skip: tmp_mnt
    +skip: core
<< /tmp >>
    skip: .* *
<< /export/swap >>
    swapasm: .
<< /nsr >>
    allow
<< /nsr/logs >>
    logasm: .
<< /var >>
    logasm: .
<< /usr/adm >>
    logasm: .
<< /usr/spool >>
    logasm: .
<< /usr/spool/mail >>
    mailasm: .
<< /usr/mail >>
    mailasm: .
```

You can tell immediately by this that there is a particular format for directives, being:

```
<< path >>
[+]instruction: pattern
```

For instance, the *Unix Standard Directives* above cite the following:

```
<< /tmp >>
skip: .* *
```

This means nothing within the **/tmp** directory will be backed up to any host for which the *Unix Standard Directives* directive has been applied to it.

The example directives I've shown has two skips in the first section, that being:

```
<< / >>
skip: tmp_mnt
+skip: core
```

The plus sign significantly alters the interpretation of the 'skip' statement:

- For the first statement, "skip: tmp_mnt", NetWorker is being instructed to skip anything in the root directory of a server called "tmp_mnt".
- For the second statement, "+skip: core", NetWorker is being instructed to skip anything in the root directory *and all of its subdirectories* called "core".

You may note in both of those scenarios I said "anything" rather than "any file". Directives apply to anything that matches, file or directory. For this reason, you have to be particularly careful about how and where you apply directives. For instance, applying the "Unix Standard Directives" on a

Linux host may seem acceptable unless you consider that the Linux kernel source (/usr/src/kernels) is just full of directories called *core*:

```
[root@orilla ~]# find /usr/src -name core -type d -print
/usr/src/kernels/2.6.32-504.el6.x86_64/drivers/memstick/core
/usr/src/kernels/2.6.32-504.el6.x86_64/drivers/infiniband/core
/usr/src/kernels/2.6.32-504.el6.x86_64/drivers/net/mlx5/core
/usr/src/kernels/2.6.32-504.el6.x86_64/drivers/usb/core
/usr/src/kernels/2.6.32-504.el6.x86_64/drivers/mmc/core
/usr/src/kernels/2.6.32-504.el6.x86_64/net/core
/usr/src/kernels/2.6.32-504.el6.x86_64/include/config/serial/core
/usr/src/kernels/2.6.32-504.el6.x86_64/include/config/core
/usr/src/kernels/2.6.32-504.el6.x86_64/include/config/sound/oss/core
/usr/src/kernels/2.6.32-504.el6.x86_64/sound/core
/usr/src/kernels/2.6.32-504.el6.x86_64/sound/aoa/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/drivers/memstick/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/drivers/infiniband/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/drivers/net/mlx5/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/drivers/usb/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/drivers/mmc/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/net/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/include/config/serial/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/include/config/core
/usr/src/kernels/2.6.32-504.1.3.el6.x86_64/include/config/sound/oss/core
```

Figure 201: 'core' directories on Linux servers

If the *Unix Standard Directives* were to be applied to a Linux server, a noticeable chunk of the kernel source directory structure would not be backed up.

In NetWorker, the *instruction* part of the directive (i.e., “what to do”) is referred to as an ASM – an Application Specific Module. These should not be confused with NetWorker Application Modules. They in fact refer to modules within the *uasmm* utility, which is the base level command called by the NetWorker *save* process in order to do a filesystem backup.

Particularly pertinent *asms* that are available for use within directives include³⁰:

Table 7: Most common options in directives

ASM	Purpose
always	Always back up a file, regardless of whether it has changed or not.
atimeasm	When backing up a file, do not change the access time for the file.
compressasm	Perform a basic compression on the data.
logasm	Used against log files that may be backed up; this prevents NetWorker reporting any changes to the file during the backup process
mailasm	Useful for Unix/Linux servers only, this performs basic mail utility file locking for default mail systems.
null	Do not backup the patterns that match but equally don't 'blank out' the parent directory in the indices if nothing is going to be backed up.
skip	Do not backup patterns that match, and do <i>not</i> include the parent directory in the indices if nothing is going to be backed up.

³⁰ These are not intended to be comprehensive, but simply to cite the ones more commonly used.

23.2 Placement

Directives may be stored on either within the NetWorker server's configuration and *applied* to clients, or they may be stored as plain text files on clients, within any directory, so long as the content of the directive applies to that directory or a subdirectory.

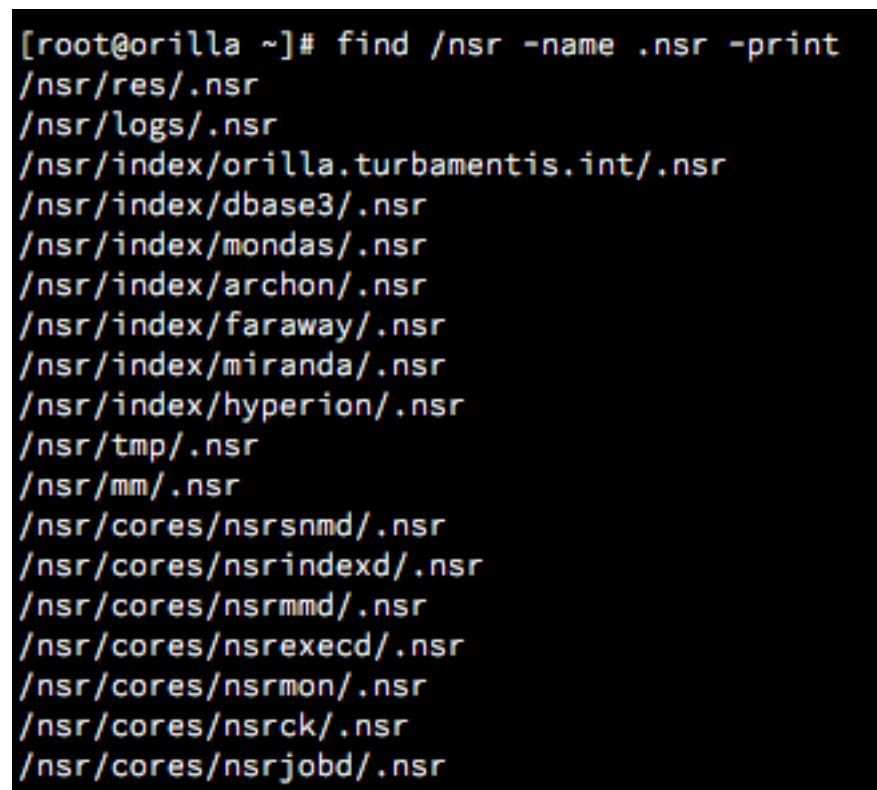
When stored on a client, directives use the following filename conventions:

- Unix/Linux/Mac OS X: **.nsr**
- Windows: **nsr.dir**

Note the preliminary dot in the **.nsr** filename for Unix/Linux/Mac OS X. This is essential.

Unless you have very specific reasons, you should always define directives within the NetWorker server configuration and apply them to clients. This allows backup administrator control over the directives.

A default NetWorker install will create *some* client-side directives, typically within the *nsr* directory itself. For instance:



```
[root@orilla ~]# find /nsr -name .nsr -print
/nsr/res/.nsr
/nsr/logs/.nsr
/nsr/index/orilla.turbamentis.int/.nsr
/nsr/index/dbase3/.nsr
/nsr/index/mondas/.nsr
/nsr/index/archon/.nsr
/nsr/index/faraway/.nsr
/nsr/index/miranda/.nsr
/nsr/index/hyperion/.nsr
/nsr/tmp/.nsr
/nsr/mm/.nsr
/nsr/cores/nsrsnmd/.nsr
/nsr/cores/nsrindexd/.nsr
/nsr/cores/nsrmmd/.nsr
/nsr/cores/nsrexecd/.nsr
/nsr/cores/nsrmon/.nsr
/nsr/cores/nsrck/.nsr
/nsr/cores/nsrjobd/.nsr
```

Figure 202: Directives created automatically by NetWorker

Feel free to examine any of the directive files established by NetWorker on a server, but be certain not to modify them or you may cause problems with critical backup and recovery options such as bootstrap (disaster) recoveries. For instance, the **/nsr/res/.nsr** file contains the following details:

```
logasm: *
```

You'll note that for the client-side directive, the path has not been included. If the path isn't included client-side, it's assumed to apply to the directory the directive is stored in. If the path is included, it must be:

- Either the path for the directory the directive file is stored in, *or*
- The path for a subdirectory of the directory the directive is stored in.

Note that if any subdirectory is a *mount point*, the client side directive may not be automatically applied from that point. (This limitation does not apply for server-side directives.)

23.3 Directive Examples

For this section, we'll cover a few different examples of directives.

23.3.1 Scenario: Skipping Database files on Microsoft SQL Server

Consider a Microsoft SQL Server host where the active database data and log files are stored in `D:\Databases`, but the NetWorker module for Microsoft Applications is used for database backups.

In this scenario, there's no point trying to back these files up as part of the filesystem backup – if VSS is fully integrated you might pick up a backup, but the files won't be fully consistent with one another and therefore unable to be used for recovery purposes. If VSS isn't fully integrated, it may just trigger a lot of errors about files being inaccessible for backup.

We can create a directive for this that looks like the following:

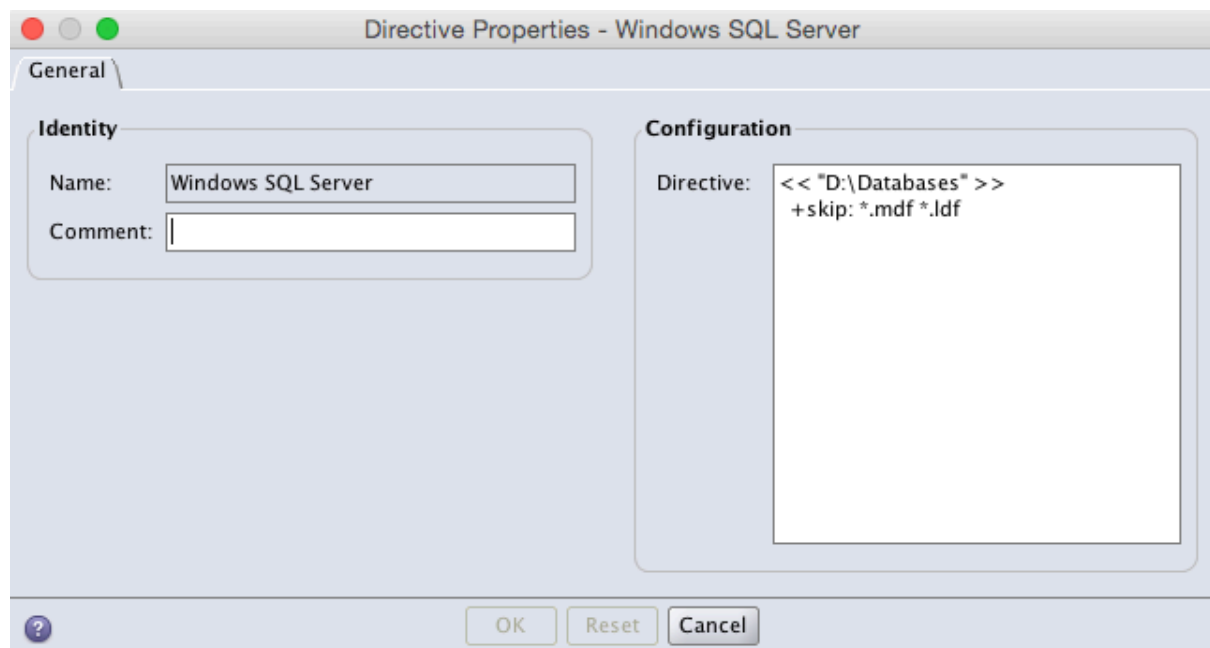


Figure 203: A directive for Microsoft SQL Databases

There's a good reason why you create directives in NMC rather than say, using `nsradmin`. If we look at resource for this directive you'll see why:

```
[root@orilla ~]# nsradmin
NetWorker administration program.
Use the "help" command for help, "visual" for full-screen mode.
nsradmin> print type: NSR directive; name: Windows SQL Server
               type: NSR directive;
               name: Windows SQL Server;
               comment: ;
               directive: "<< \"D:\\Databases\" >>
               +skip: *.mdf *.ldf";
nsradmin>
```

Figure 204: How directives appear in `nsradmin`

Even Unix directives become ugly and cumbersome to work with in nsradmin quite quickly, but Windows directives, where quotes, backslashes and colons may be required become very painful. It's best to leave them to NMC.

Once the directive is created, it can then be applied the client instance performing *filesystem* backups.

23.3.2 Example: Skipping Multimedia Content

It may be that corporate policy is to not backup any multimedia content on a corporate fileserver. It's determined that the most likely files that could end up in this bucket have the following extensions:

- .mp3
- .mp4
- .m4a
- .m4v
- .mov
- .wav
- .aiff
- .avi

In this case, the business requires that any file with that extension is not backed up, no matter where it is on the fileserver. The directives for this would appear as follows:

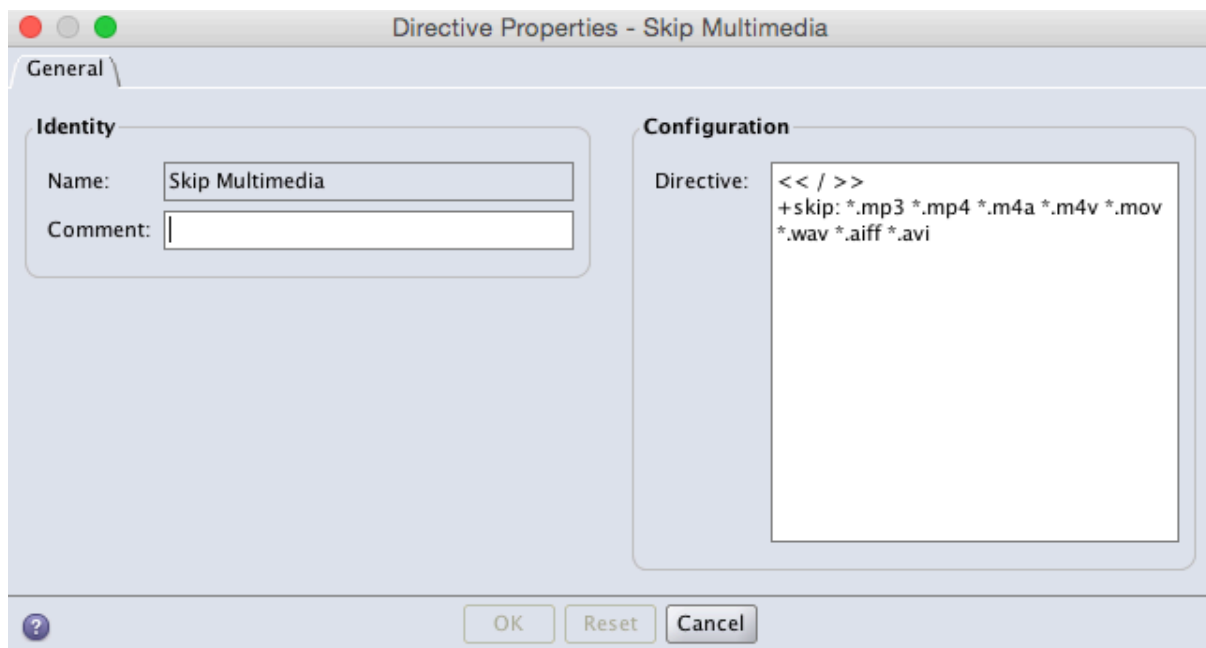


Figure 205: Skipping multimedia content

Note:

The list of matching paths *does not* extend over multiple lines (i.e., there's no carriage returns).

An alternate way of specifying this directive would be:

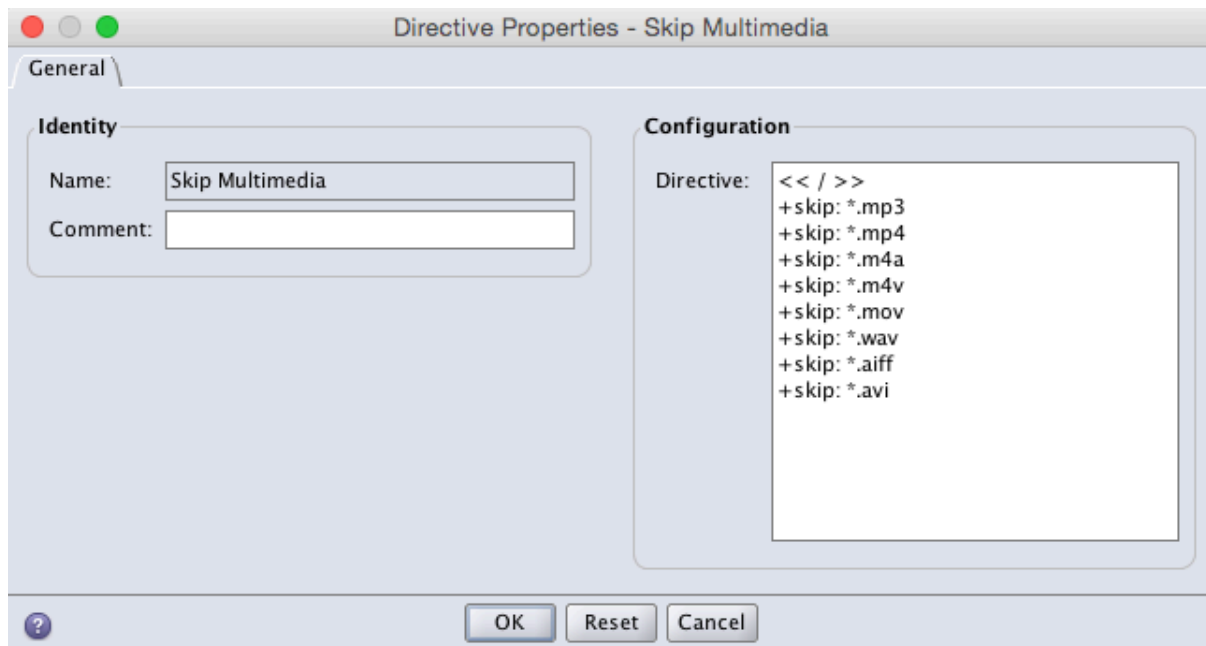


Figure 206: Alternate format for long directive lines

You may think this directive can only apply to a Unix system, since it specifies a path of “/”. That’s not technically true. The “/” path, if specified in a directive applied to a Windows system will apply to *all* drives and mount points. If you want to confirm that, consider a recovery session on Windows:

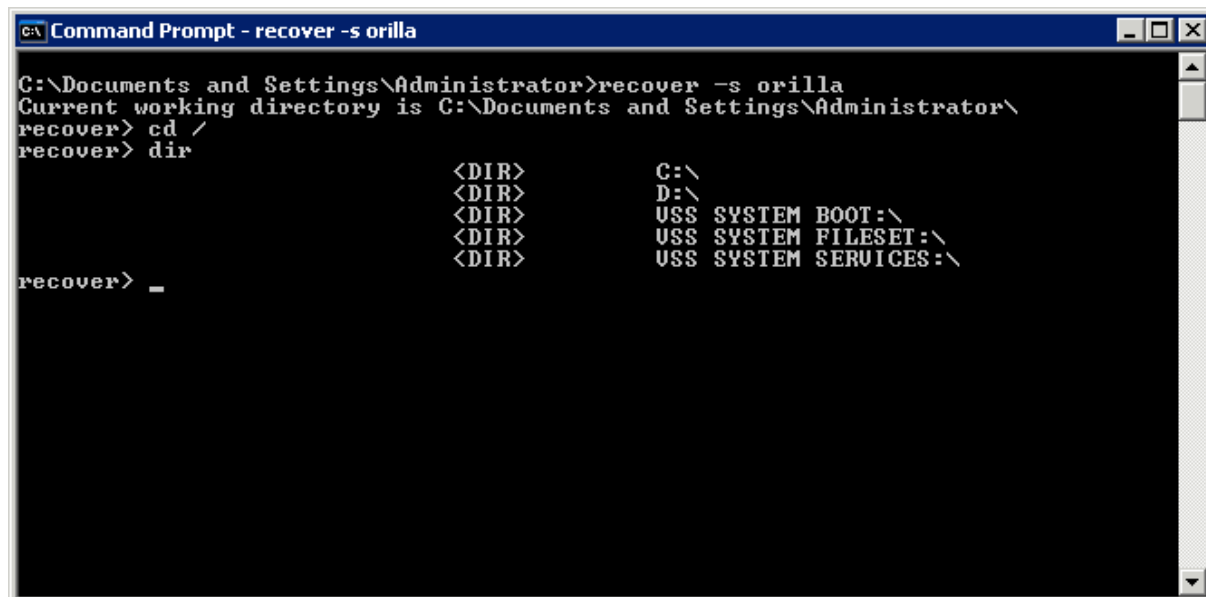


Figure 207: Windows recovery session in the “/” path.

It should be noted that while the special savesets (SYSTEM STATE, etc.) appear at this level, directives on Windows applied to the “/” path will not be applied to them. The directives are solely applied to standard filesystems.

23.3.3 Example: Split Backups of a very large filesystem

For our final scenario, we’re going to consider an increasingly rare situation (given options such as parallel save streams and block level backup), but one which is worth understanding, and that’s when a filesystem is too large to perform a full backup in one session.

Consider a Unix system (we'll unimaginatively call it 'fileservr') that hosts a corporate file share:

```
/fileshare
```

Underneath that directory, there's a set of directories for each of the departments within the organisation. For instance:

```
/fileshare/common  
/fileshare/consulting  
/fileshare/engineering  
/fileshare/finance  
/fileshare/human-resources  
/fileshare/sales
```

In this scenario, it may be that several of the subdirectories of /fileshare are so large that if an attempt to do a full backup on all of them at once were performed, it would take too long for the backup to complete.

To get around the problem, the backup administrator might define the following client instances. We'll limit ourselves just to Daily groups to start with:

Client Name	Save Set(s)	Daily Group	Schedule
fileservr	All	Daily OS	Full Friday/Incr Rest
fileservr	/fileshare/common	Daily Fileservr	Full Friday/Incr Rest
fileservr	/fileshare/consulting /fileshare/engineering	Daily Fileservr	Full Saturday/Incr Rest
fileservr	/fileshare/finance /fileshare/human-resources	Daily Fileservr	Full Tuesday/Incr Rest
fileservr	/fileshare/sales	Daily Fileservr	Full Sunday/Incr Rest

In this scenario, the "Daily Fileservr" group would *not* have a level or schedule assigned to it, meaning the schedules would be assigned to the individual client instances for 'fileservr' when running. That way:

- **Friday:**
 - All regular filesystems on 'fileservr' get a full backup
 - /fileshare/common gets a full backup
 - All other /fileshare/* directories get an incremental backup
- **Saturday:**
 - All regular filesystems on 'fileservr' get an incremental backup
 - /fileshare/consulting and /fileshare/engineering get a full backup
 - All other /fileshare/* directories get an incremental backup
- **Sunday:**
 - All regular filesystems on 'fileservr' get an incremental backup
 - /fileshare/sales gets a full backup
 - All other /fileshare/* directories get an incremental backup
- **Tuesday:**
 - All regular filesystems on 'fileservr' get an incremental backup
 - /fileshare/finance and /fileshare/human-resources get a full backup
 - All other /fileshare/* directories get an incremental backup

There's a seeming hole in that logic though – the 'All' save set would typically pick up all the subdirectories of /fileshare. To avoid that, there are two options:

- Change the 'All' save set to an explicit list of all the other non-/fileshare savesets on the host

- Configure directives for the 'All' client instance to exclude the /fileshare directory and all its subdirectories.

The first option is reckless and not recommended for the simple reason that if another filesystem is added to the server and the client instance is not updated, that filesystem will not get backed up³¹.

Based on the previous list of ASMs, you may think there are two options for excluding the /fileshare contents from the 'All' client instance. Technically you'd be right, but only one option is practically correct.

The *incorrect* method would be to use the skip directive for the 'All' client, viz.:

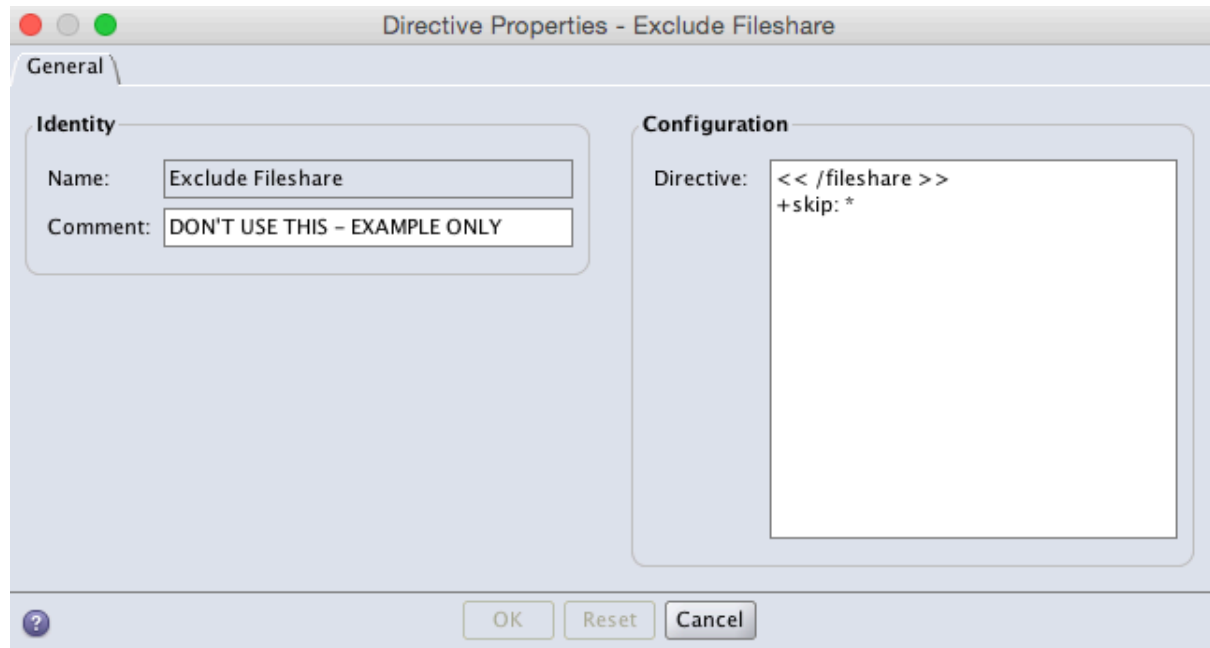


Figure 208: Where *not* to use the skip directive

The reason you wouldn't use the skip directive is based on the previous explanation of it:

“Do not backup patterns that match, and *do not* include the parent directory in the indices if nothing is going to be backed up.”

If you used this directive against the 'All' client instance, here's what would happen:

Any time the client instance with the 'All' save set was run, the index information for the client *for that point in time* would exclude the contents of the /fileshare directory. That means you'd have to execute two recoveries to recover the entire server – one for everything else, then one for /fileshare. You'll also have to be able to pinpoint for recovery browsing those times when an appropriate /fileshare/subdirectory backup had completed, but an 'All' style backup had not started.

For example, if you executed backups as follows with the *skip* option:

1. 21:00 – Backup of /fileshare/common
2. 22:55 – Backup of /fileshare/consulting and /fileshare/engineering
3. 23:55 – Backup of 'All' instance

A recovery browse operation executed after backup (3) had completed would show the /fileshare directory as being *empty*.

³¹ I have seen many instances of the years where this has precisely happened. Sometimes filesystems have gone *months* if not *years* without getting backed up.

Instead of using the *skip* option, the only appropriate, recovery compatible option is the *null* option:

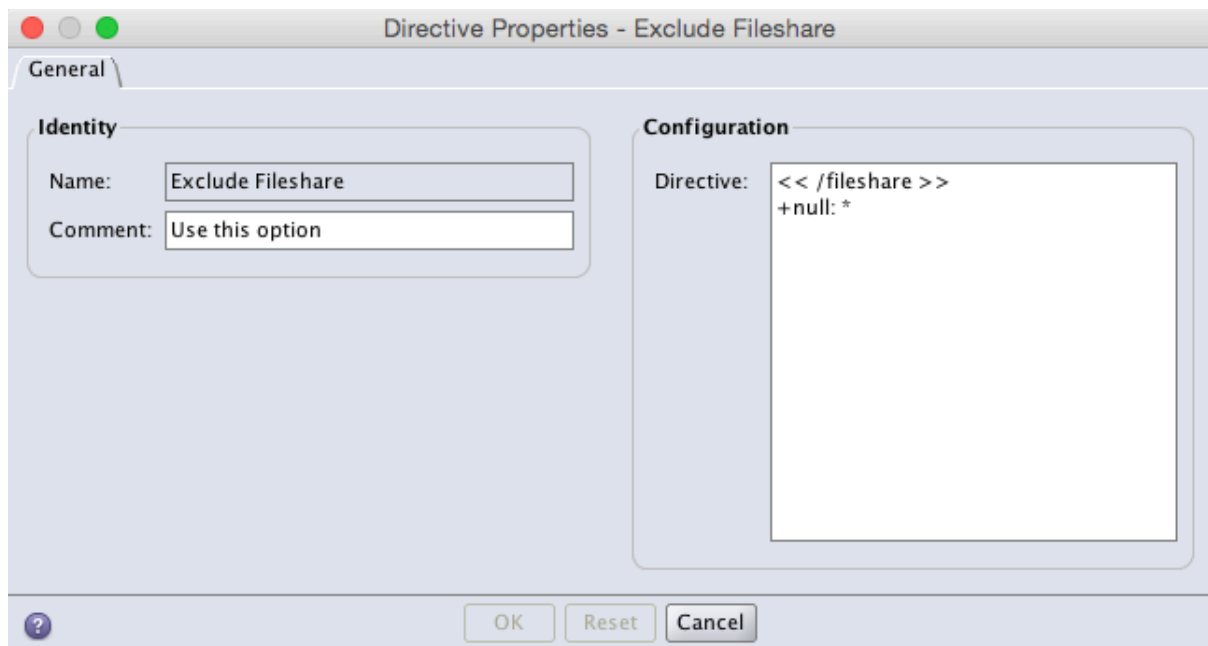


Figure 209: Correct use of the *null* directive option

By comparison to the *skip* option, when the *null* option is run against the */fileshare* directory, NetWorker will keep index references *in this backup* to previously completed backups. Thus, you could backup in any order without fear of surprise when a recovery is executed. For example:

1. 21:00 – Backup of */fileshare/common*
2. 22:55 – Backup of */fileshare/consulting* and */fileshare/engineering*
3. 23:55 – Backup of 'All' instance

A recovery browse operation executed after backup (3) had completed would show the contents of the */fileshare* directory as of the time those subdirectories were most recently backed up.

Wrapping up

EMC NetWorker is an incredibly advanced enterprise backup product that has one of the most important components required in enterprise software: an extensive command line interface.

Making use of NetWorker's CLI, you can automate activities, generate custom reports, extend control options and take control during serious challenges.

Practice makes perfect, however: the NetWorker CLI is best grasped by regularly using it. That's why having a lab environment you can practice in is absolutely critical to becoming a true NetWorker power user. It doesn't matter if the environment is entirely virtualised – it just matters that you use it, and practice with it before diving into your production environment.

As for more involved backup control options (such as pre and post processing, and directives), practice still makes perfect. Experiment with these options when you can so you can better understand how to achieve your backup and recovery requirements most efficiently.

And remember: always have backups.

Further Reading

In addition to keeping a nearby copy of the official NetWorker documentation, you may want to consider the following articles on the NetWorker Blog for expanded information to some of the topics covered in this guide, or to provide additional information about more recent versions of NetWorker:

Table 8: Recommended blog articles

Topic	Link
Bypassing NetWorker for media movement	http://nsrd.info/blog/2010/03/15/basics-bypassing-networker-for-media-movement/
Checkpoint your backups	http://nsrd.info/blog/2013/11/07/checkpoint-your-backups/
Client side compression gets a squeeze	http://nsrd.info/blog/2012/07/31/client-side-compression-gets-a-squeeze/
Debugging nsrmmd	http://nsrd.info/blog/2010/10/28/debugging-nsrmmd/
Fixing NSR Peer Information Errors	http://nsrd.info/blog/2009/02/23/basics-fixing-nsr-peer-information-errors/
Introducing NetWorker 8	http://nsrd.info/blog/2012/07/11/introducing-networker-8/

Topic	Link
Learning NetWorker	http://nsrd.info/blog/2009/01/25/learning-networker/
LinuxVTL and NetWorker	http://nsrd.info/blog/2010/10/14/new-micromanual-linuxvtl-and-networker/
mminfo – savetime and greater than/less than	http://nsrd.info/blog/2009/02/01/basics-mminfo-savetime/
mminfo and NOT queries	http://nsrd.info/blog/2010/10/19/mminfo-and-not-queries/
New NetWorker Technical Documents	http://nsrd.info/blog/2011/05/05/new-networker-technical-documents/
Recovering with scanner and uasm	http://nsrd.info/blog/2009/04/22/recovering-with-scanner-and-uasm/
Setting cleaning tape usage/registering a cleaning tape	http://nsrd.info/blog/2009/03/09/basics-setting-cleaning-tape-usageregistering-a-cleaning-tape/
Understanding skip vs null directives in detail	http://nsrd.info/blog/2009/07/08/basics-null-vs-skip-directives/
What's new in 8.2?	http://nsrd.info/blog/2014/06/30/whats-new-in-8-2/

Indices

Table of Figures

Figure 1: mminfo default output (24 hours backups)	11
Figure 2: Specifying an alternate order output for mminfo.....	12
Figure 3: Using a query and sort order.....	13
Figure 4: Querying on multiple clients	16
Figure 5: Query based on a date range	17
Figure 6: Narrowing down a query	18
Figure 7: mminfo query run from Windows	18
Figure 8: Example of mminfo queries using single quotes on Windows.....	19
Figure 9: mminfo -m output (disk volumes).....	19
Figure 10: mminfo -m output (tape volumes).....	20
Figure 11: mminfo -mv (verbose media report) output	20
Figure 12: Specifying report columns in mminfo.....	21
Figure 13: Specifying column width in report output	21
Figure 14: Getting the long-form saveset ID	22
Figure 15: Finding files on adv_file devices based on the long-form saveset ID	22
Figure 16: Using multiple width fields in a custom report	24
Figure 17: Producing a volume aging report	25
Figure 18: mminfo output featuring VBA virtual machine backups	25
Figure 19: mminfo's new VBA specific report output	26
Figure 20: Reporting using the 'vmname' and 'backup_size' options	26
Figure 21: Arbitrary field separation in mminfo reports.....	27
Figure 22: Producing comma-separated output from mminfo	27
Figure 23: XML mminfo output.....	28
Figure 24: Actual XML data content in mminfo XML output.....	29
Figure 25: Very verbose mminfo output	30
Figure 26: Sample volume-order output	33
Figure 27: Sample volume-order verbose output.....	33
Figure 28: Sample execution of custom Perl script, 'mminfo2html'	34
Figure 29: Sample output from mminfo2html.....	34
Figure 30: nsrinfo, a first look	36
Figure 31: Isolating saveset times in nsrinfo (1)	37
Figure 32: Isolating saveset times in nsrinfo (2)	38

Figure 33: Isolating saveset times in nsrinfo (3).....	38
Figure 34: Identifying backup versions via recover	39
Figure 35: Identifying backup versions via nsrinfo.....	39
Figure 36: Using nsrinfo with the verbose flags	39
Figure 37: Using nsrinfo to view all files in a client index	40
Figure 38: Using nsrinfo on Unix to search for backed up files.....	40
Figure 39: Using nsrinfo on Windows to search for backed up files	40
Figure 40: Running gstclreport without an accessible Java environment.....	42
Figure 41: Running gstclreport with JAVA_HOME correctly established.....	42
Figure 42: Generating a basic report out of gstclreports	43
Figure 43: Determining gstclreport configuration options	44
Figure 44: Generating a report with gstclreport and custom configuration options.....	45
Figure 45: Dealing with date ranges in gstclreport (1).....	45
Figure 46: Dealing with date ranges in gstclreport (2).....	46
Figure 47: Dealing with date ranges in gstclreport (3)	47
Figure 48: Dealing with start dates only in gstclreport	47
Figure 49: Using nsr_render_log for a specific client and date/time range	49
Figure 50: Rendering a log on a system without the originating module	50
Figure 51: Configuring the runtime rendered log for a raw file	51
Figure 52: Options for nsrsgrpcomp	52
Figure 53: Extracting the most recent savegroup completion details with nsrsgrpcomp	52
Figure 54: Listing savegroup completion information for a specific group	52
Figure 55: Accessing details of a specific savegroup execution with nsrsgrpcomp.....	53
Figure 56: Obtaining a list of all available savegroup completion reports	53
Figure 57: Accessing details of a specific client and group	54
Figure 58: Running nsrsgrpcomp from a host other than the NetWorker server	54
Figure 59: Extracting summary information from nsrsgrpcomp	55
Figure 60: Refining nsrsgrpcomp summary output, by client	55
Figure 61: nsrwatch on a Unix platform	56
Figure 62: Running nsrwatch on Windows	57
Figure 63: More devices than nsrwatch will show	57
Figure 64: nsrwatch with limited screen space	58
Figure 65: Tabbing between different sections of nsrwatch	58

Figure 66: Viewing only mounted devices in nsrwatch	59
Figure 67: Default execution of nsrmm	61
Figure 68: nsrmm output showing combined tape/disk device status.....	61
Figure 69: Unmounting and mounting volumes with nsrmm	62
Figure 70: Mounting a volume in read-only mode with nsrmm	62
Figure 71: 'Remounting' a read-only volume as read-write	63
Figure 72: Labelling a volume using nsrmm.....	63
Figure 73: Labelling a volume without mounting it using nsrmm.....	64
Figure 74: Relabeling a volume with nsrmm.....	64
Figure 75: Automatically answering 'yes' to nsrmm prompts (dangerous).....	64
Figure 76: Labelling a previously labelled volume into a new pool	65
Figure 77: Flagging a volume as being offsite.....	67
Figure 78: Marking a volume as recyclable.....	67
Figure 79: Saveset marked as recyclable via a volume being marked as recyclable.....	68
Figure 80: mminfo output showing browse and retention time for a saveset	69
Figure 81: Changing the browse and retention time for a saveset.....	69
Figure 82: Standard nsrjb output.....	70
Figure 83: Invoking nsrjb without options in a multi-jukebox environment	71
Figure 84: Invoking nsrjb against a specific jukebox	71
Figure 85: Using nsrjb with the verbose flag	72
Figure 86: Running a fast inventory operation	73
Figure 87: Using fast inventory when volumes have not previously been labelled.....	73
Figure 88: Using nsrjb with higher levels of verbosity	74
Figure 89: Performing a slow inventory using extended verbose mode	74
Figure 90: Limiting a jukebox inventory to specific slots.....	75
Figure 91: Limiting jukebox operations to a particular device	76
Figure 92: Basic jukebox reset command	76
Figure 93: Jukebox reset command when there are volumes to unload	77
Figure 94: Performing a media label operation.....	77
Figure 95: Viewing the status of the jukebox after a label operation	78
Figure 96: Recycling a volume into another pool	78
Figure 97: Using the recycle option against a recyclable volume.....	79
Figure 98: Performing a volume load operation	80

Figure 99: Performing a volume unload operation, by volume name	80
Figure 100: Loading a volume without mounting it	81
Figure 101: Withdrawing a volume into the CAP	82
Figure 102: Jukebox state after a withdraw operation	82
Figure 103: Performing a deposit operation	83
Figure 104: Standard inquire output	84
Figure 105: The inquire command with persistent device names	84
Figure 106: Output from sjisn	85
Figure 107: Inquire output showing second jukebox	86
Figure 108: Output from sjirdtag	86
Figure 109: Noting changes to reported information in sjirdtag	87
Figure 110: Using sjimm	88
Figure 111: NMC view of a NetWorker resource	89
Figure 112: NetWorker resource database as files and directories	90
Figure 113: NetWorker resource in plain text	90
Figure 114: Command line options for nsradmin	92
Figure 115: Running nsradmin on a client without referencing a server	92
Figure 116: Getting help from nsradmin	93
Figure 117: Getting help on individual commands with nsradmin	93
Figure 118: Determining valid configuration types in nsradmin	94
Figure 119: Viewing types for nsradmin against the client program	94
Figure 120: nsradmin help for the 'print' command	95
Figure 121: Displaying all policies on a NetWorker server	96
Figure 122: Viewing just policy names	97
Figure 123: Viewing a single policy in nsradmin	97
Figure 124: Turning off display restrictions and re-printing a query	98
Figure 125: nsradmin 'option' command	98
Figure 126: Viewing hidden details of NetWorker resources using the option command	99
Figure 127: Starting a group from within nsradmin	101
Figure 128: Setting the level of a group to 'full' always	102
Figure 129: Starting and then stopping a group	102
Figure 130: Checking the status of a group	103
Figure 131: Understanding the work list	103

Figure 132: Turning cloning on for a group	104
Figure 133: Group status while cloning	105
Figure 134: Using append instead of update	106
Figure 135: Specifying Windows save sets in nsradmin	106
Figure 136: The 'Month' policy	107
Figure 137: Viewing the different period types available to NetWorker policies.....	108
Figure 138: Creating Daily and Monthly policies.....	108
Figure 139: Viewing the newly created Daily and Monthly policies	109
Figure 140: Settings for the Default schedule	109
Figure 141: Creating the Daily schedule.....	111
Figure 142: Creating the Monthly schedule.....	112
Figure 143: Daily schedule as shown by NMC	112
Figure 144: Monthly schedule as shown by NMC.....	113
Figure 145: Viewing the Default group.....	114
Figure 146: Creating the Daily group.....	115
Figure 147: Creating the Monthly group.....	115
Figure 148: Viewing an existing client instance	116
Figure 149: Creating a new client in nsradmin	117
Figure 150: Second client create command in nsradmin	117
Figure 151: Viewing the Default pool in nsradmin.....	118
Figure 152: Creating the Daily pool in nsradmin.....	119
Figure 153: Creating the Daily Clone pool.....	120
Figure 154: Monthly pool.....	120
Figure 155: Monthly Clone pool	121
Figure 156: Configuring the Daily and Monthly groups to clone	122
Figure 157: Viewing a device in nsradmin	123
Figure 158: Viewing active devices using nsradmin	124
Figure 159: Deleting clients using nsradmin.....	125
Figure 160: Turning cloning off for groups.....	126
Figure 161: Deleting pools in nsradmin.....	127
Figure 162: Deleting the groups, schedules and policies.....	128
Figure 163: Bulk addition of clients to NetWorker using nsradmin.....	130
Figure 164: Performing a bulk delete in nsradmin.....	131

Figure 165: Running the create-policy.bat script	132
Figure 166: Script to create a new policy on using Perl	133
Figure 167: Bulk creation of resources to be used for scripting	135
Figure 168: Bulk creation of resources on Windows	135
Figure 169: New client script being executed on Linux	136
Figure 170: Executing the create-client script on Windows	137
Figure 171: Using nsradmin to view the NSRLA resource on a client	138
Figure 172: Viewing the peer certificate information for a client.....	139
Figure 173: Using regular expressions to show clients in nsradmin	140
Figure 174: How NetWorker interprets regular expressions when it isn't expecting them	140
Figure 175: Limitations with regular expressions in nsradmin.....	141
Figure 176: Running nsradmin in offline mode on Linux/Unix.....	142
Figure 177: Running nsradmin in offline mode on Windows	142
Figure 178: Standard nsrck output, starting in NetWorker 8.2	144
Figure 179: NetWorker daemon log content from running nsrck -m	144
Figure 180: Using the nsrls command against the media database	145
Figure 181: Performing a basic index check	145
Figure 182: Performing a basic client file index listing/summary	146
Figure 183: Executing an nsrck -L3	147
Figure 184: Performing an index rebuild against a client file index	147
Figure 185: Performing an index recovery.....	147
Figure 186: Default output of dbgcommand.....	151
Figure 187: ps output for nsrmmd processes	152
Figure 188: Generating device information using dbgcommand	152
Figure 189: Using the FlushDnsCache option for dbgcommand	154
Figure 190: Turning on debug mode level 9	155
Figure 191: Debug log information	155
Figure 192: Turning debug mode off.....	155
Figure 193: Log file in use	156
Figure 194: Specifying pre and/or post commands in NetWorker 8.2+	159
Figure 195: Configuring a client pre command.....	160
Figure 196: Results of pre command	161
Figure 197: Recovering a file transferred as part of a pre command.....	161

Figure 198: Savegroup completion report showing successful pre command execution	162
Figure 199: Pre command failure	162
Figure 200: Post command failure	162
Figure 201: 'core' directories on Linux servers	164
Figure 202: Directives created automatically by NetWorker	165
Figure 203: A directive for Microsoft SQL Databases.....	166
Figure 204: How directives appear in nsradmin	166
Figure 205: Skipping multimedia content.....	167
Figure 206: Alternate format for long directive lines.....	168
Figure 207: Windows recovery session in the "/" path.	168
Figure 208: Where <i>not</i> to use the skip directive.....	170
Figure 209: Correct use of the <i>null</i> directive option	171

Index of Tables

Table 1: Ordering options in mminfo	12
Table 2: Additional report fields	23
Table 3: Display toggle options in nsrwatch.....	58
Table 4: Options for limiting the extent of an nsrjb operation to particular volumes, slots or devices	75
Table 5: Standard nsrck check levels.....	146
Table 6: Scenarios for running nsrck index rebuilds.....	148
Table 7: Most common options in directives.....	164
Table 8: Recommended blog articles	173