GRF Reference Guide 1.4

Ascend Communications, Inc.

Part Number: 7820-2012-001

For software version 1.4

December, 1997

Ascend Customer Service

You can request assistance or additional information by telephone, email, fax, or modem, or over the Internet.

Obtaining Technical Assistance

If you need technical assistance, first gather the information that Ascend Customer Service will need for diagnosing your problem. Then select the most convenient method of contacting Ascend Customer Service.

Information you will need

Before contacting Ascend Customer Service, gather the following information:

- Product name and model
- Software and hardware options
- Software version
- Service Profile Identifiers (SPIDs) associated with your product
- Whether you are routing or bridging with your Ascend product
- Type of computer you are using
- Description of the problem

How to contact Ascend Customer Service

After you gather the necessary information, contact Ascend in one of the following ways:

Telephone in the United States	800-ASCEND-4 (800-272-3634)
Telephone outside the United States	510-769-8027 (800-697-4772)
Austria/Germany/Switzerland	(+33) 492 96 5672
Benelux	(+33) 492 96 5674
France	(+33) 492 96 5673
Italy	(+33) 492 96 5676
Japan	(+81) 3 5325 7397
Middle East/Africa	(+33) 492 96 5679
Scandinavia	(+33) 492 96 5677
Spain/Portugal	(+33) 492 96 5675
UK	(+33) 492 96 5671
Email	support@ascend.com
Email (outside US)	EMEAsupport@ascend.com
Facsimile (FAX)	510-814-2312
Customer Support BBS by modem	510-814-2302

You can also contact the Ascend main office by dialing 510-769-6001, or you can write to Ascend at the following address:

Ascend Communications
1701 Harbor Bay Parkway
Alameda, CA 94502

Need information about new features and products?

Ascend is committed to constant product improvement. You can find out about new features and other improvements as follows:

- For the latest information about the Ascend product line, visit our site on the World Wide Web:

  `http://www.ascend.com`
- For software upgrades, release notes, and addenda to this manual, visit our FTP site:

  `ftp.ascend.com`

Contents

Figures

Tables

About This Guide

This guide provides usage information about system management commands and GateD configuration statements for network administrators installing and configuring the GRF.

Unless otherwise noted, information in this Guide applies to GRF 400, GRF 1600, and GR-II systems using RMS nodes.

How to use this guide

This guide contains two chapters:

- Chapter 1, "System Commands," describes the set of GRF-specific and UNIX-like commands that monitor and manage system, memory, and interface functions.
- Chapter 2, "GateD Configuration Statements," details the configurable attributes and options for Ascend version 3.5b3 of GateD.
- Appendix A, "Configuration File Templates," contains the templates for all GRF `/etc/xxx.conf` configuration files.
- Appendix B, "Warranty," contains the product warranty information.

This guide also includes an index.

What you should know

Configuring and monitoring the GRF requires that a Network Administrator have experience with and an understanding of UNIX systems, and the ability to navigate in a UNIX environment. Knowledge of UNIX, its tools, utilities, and editors is useful, as is experience with administering and maintaining a UNIX system.

Configuring the GRF requires network experience and familiarity with:

- UNIX systems and commands
- IP protocol and routing operations
- IP internetworking

The Network Administrator must understand how TCP/IP internetworks are assembled; what interconnections represent legal topologies; how networks, hosts, and routers are assigned IP addresses and configured into operation; and how to determine and specify route table (routing) information about the constructed internetwork(s). Although not required, a high-level understanding of SNMP is useful.

Documentation conventions

This section explains all the special characters and typographical conventions in this manual.

Convention	**Meaning**
`Monospace text`	Represents text that appears on your computer's screen, or that could appear on your computer's screen.
Italics	Represent variable information. Do not enter the words themselves in the command. Enter the information they represent. In ordinary text, italics are used for titles of publications, for some terms that would otherwise be in quotation marks, and to show emphasis.
[ ]	Square brackets indicate an optional argument you might add to a command. To include such an argument, type only the information inside the brackets. Do not type the brackets unless they appear in bold type.
|	Separates command choices that are mutually exclusive.
Note:	Introduces important additional information.
⚠ **Caution:**	Warns that a failure to follow the recommended procedure could result in loss of data or damage to equipment.
⚡ **Warning:**	Warns that a failure to take appropriate safety precautions could result in physical injury.

Manual set

The GRF 1.4 documentation set consists of the following manuals:

- *GRF 400/1600 Getting Started 1.4*
- *GRF Configuration Guide 1.4*
- *GRF Reference Guide 1.4* (this manual)

Related publications

Here are some related publications that you may find useful:

- *TCP/IP Network Administration*, Craig Hunt (O'Reilly & Associates, Inc.)
- *Essential System Administration,* Æleen Frisch (O'Reilly & Associates, Inc.)
- *Internetworking with TCP/IP, Volume 1*, Douglas E. Comer, David L. Stevens (Prentice-Hall)
- *TCP/IP Illustrated, Volume 1,* W. Richard Stevens (Addison-Wesley)

System Commands

1

Chapter 1 describes the GRF system commands:

Introduction to system commands

The GRF router user environment has two interfaces:

– command-line interface (CLI)
– UNIX shell

Configuration and management tasks are divided between the two interfaces, although the migration is to eventually provide all capabilities in the CLI.

Interface, media, and protocol configuration is done from the UNIX shell, using a UNIX editor to edit the `/etc/xxxxxx.conf` configuration files. The standard UNIX command set is available in the UNIX shell. UNIX commands are not described in this manual unless the command has been adapted to the GRF, **ping** is an example of such a command. GRF control board memory does not provide space to contain the entire set of UNIX manpages.

Certain system management tasks are done in the CLI. These include specifying system-wide defaults for dump and executable binary files in the Dump and Load profiles. Each slot has a Card profile in which card-specific settings can be made for individual media cards.

Chapter 1 combines the commands from both interfaces, arranges them alphabetically, and provides a desciption or definition of each. This organization makes it easy for you to find a command.

These markers tell you in which interface you can use a particular command:

(CLI)	-	only from the CLI
(CLI+shell)	-	from both CLI and UNIX shell
(grrmb)	-	only after you enter the **grrmb** command, available from both the CLI and the UNIX shell
(gsm)	-	prompt at which you enter **gsm** subcommands while in the CLI (**gsm** prompt from the shell includes host domain name)
(shell)	-	only from the UNIX shell

Systems with RMS nodes

If your system uses an RMS node and is upgraded to the current Ascend OS software release, your original commands are still usable. Certain commands operate only on newer GRF routers with an updated control board and no RMS node, those commands are described as such in this manual. If you enter one of those commands, you get a “`may only be used on GRF`” message but nothing else happens. These four commands operate only on RMS-node systems: **grinstall**, **grc**, **grms**, and **pwrfaild**.

A note about grinchd.conf

The CLI has replaced `grinchd.conf`. The first time you install 1.4 software over a prior release, all the `grinchd.conf` settings are automatically transferred to their proper places in CLI profiles.

Applying commands to hardware

The GRF 400 has four media card slots, the GRF 1600 and GR-II have 16. Because of this, some examples in this manual use slots 0–3, others use 0–15. Particular slot numbers are not significant in examples.

Slot number systems

Any of three numbering systems can be applied to the chassis slots, decimal is commonly used.

–	decimal in the form: [1-9]	=	slot 1
–	hexadecimal in the form: 0x	=	slot 0x1
–	octal in the form: 0	=	slot 01

To avoid confusion with octal, do not preface decimal numbers with 0.

GRF 400 slot numbers

Slots are numbered top to bottom as shown in Figure 1-1, the control board is always 66:

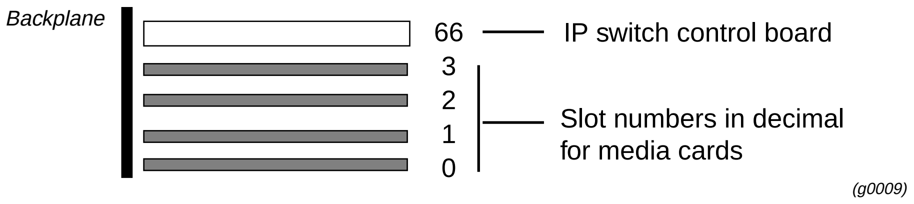


Figure 1-1. Side view of GRF 400 chassis with slots numbered

GRF 1600 slot numbers

The GRF 1600 has 16 media card slots, 0–15, a control board, and a switch board. Slots are numbered left to right as shown in Figure 1-2, the control board is always 66:

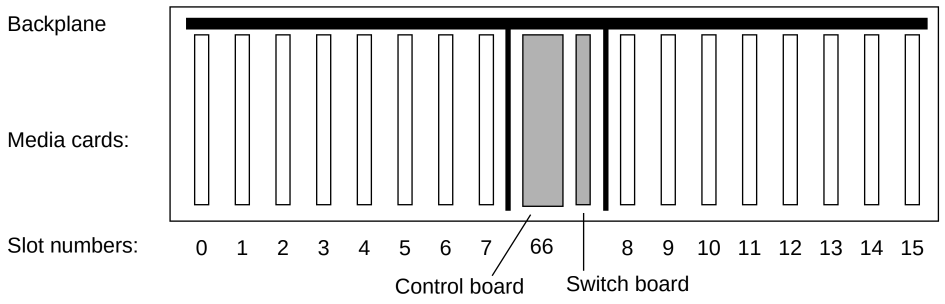


Figure 1-2. Top down view of GRF 1600 chassis with slots numbered

GR-II slot numbers

Slots are numbered left to right as shown in Figure 1-3, the router manager board is always 66:

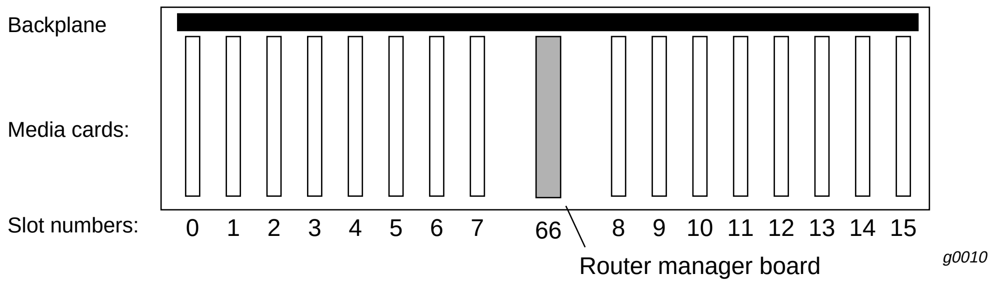


Figure 1-3. Top down view of GR-II chassis with slots numbered

Control board memory

The GRF 400 and GRF 1600 control boards have a different memory architecture from the RMS node. On the GRF systems, the RMS disk and SRAM are replaced by system RAM, internal 85MB flash, and, optionally, external PCMCIA storage devices.

To help you use the GRF memory commands, refer to Figure 1-4 and Figure 1-5 for diagrams of memory organization and components.

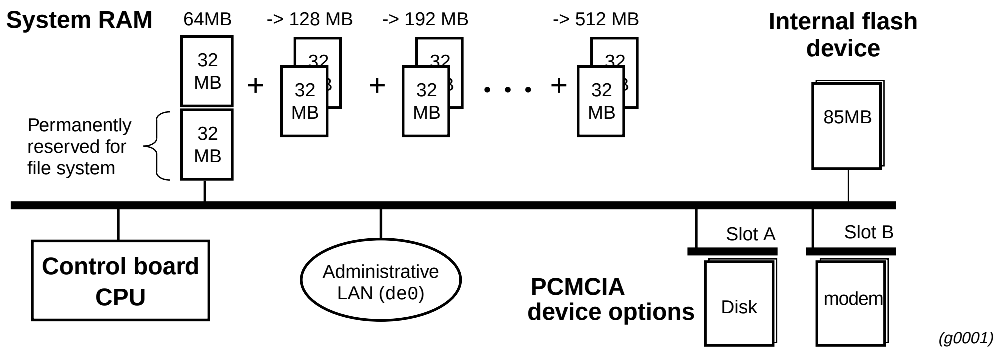


Figure 1-4. GRF memory architecture and options

RAM The GRF is equipped with a minimum of 64MB of system RAM, 32MB of which are permanently reserved for the file system, including configuration files. The remaining 32MB are used by the operating system and user applications such as GateD. System RAM can be upgraded to a maximum of 512MB.

In the CLI, use the CLI **mem** command to display the amount of system RAM installed on the GRF control board, enter:

```
super> mem ?
System memory 256 Mbytes
```

You can also determine the amount of RAM by reading the System profile, it is specified in the physical memory field:

```
super> read system
super> list
      .
      .
      .
physical-memory = 256
```

Figure 1-5 shows the area of system RAM that can be expanded to meet site requirements.

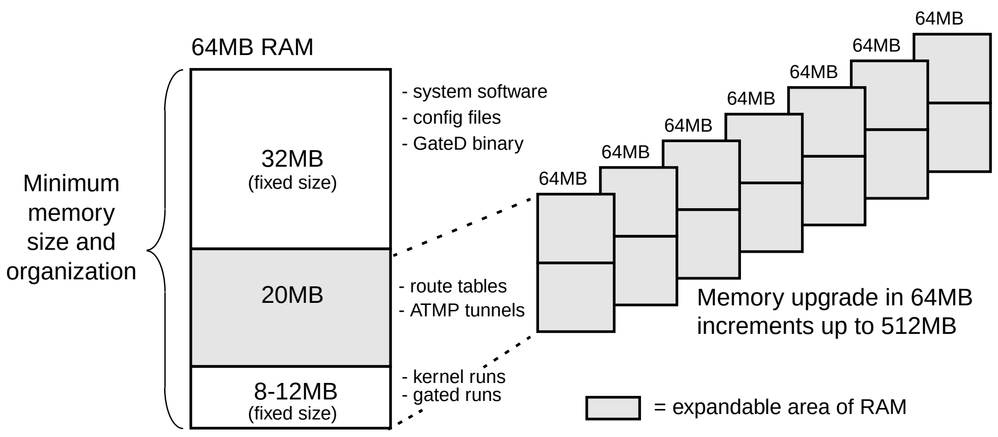


Figure 1-5. Expandable area of system memory

System memory can be upgraded in 64MB increments up to a total of 512MB, including the 32MB always reserved for the file system. To permit the software to operate in the allocated memory, certain portions of standard UNIX, such as man page files, are omitted. (Man pages for GRF commands are maintained.) Logging is configured to be done remotely via **syslog** or locally to external flash.

Internal flash The GRF has an internal 85MB ATA flash device from which the system boots. This memory is available for storing different versions of site configurations and operating software.

External flash PCMCIA slots on the control board support various sizes of ATA flash devices. Although external flash can be used to back up and share router configurations among multiple GRF systems, a GRF cannot boot from an external flash device. Local logging and dumping to external flash is also supported. The **grwrite** command writes files from system RAM to internal flash, **grsnapshot** copies files between internal and external flash devices. Commands for flash device management are discussed in the *GRF Configuration Guide*.

PCMCIA devices PCMCIA slot A is used for a portable external disk device. A PCMCIA modem device can operate in either slot A or B.

Command conventions

– In the CLI, all commands are lower case.

– In the UNIX shell, commands are case sensitive.

– Use a space to separate the command from any arguments.

– Expressions in italics are variables: *slot number*

– Expressions enclosed in brackets are optional: [*ipadd*]

grrmb command summary

The following series starts the GR##> prompt, displays a list of commands, and then exits **grrmb** to return to the CLI prompt:

```
super> grrmb
GR 66> ?
Ascend Commands
        ?
        bignore
        fan
        maint -[hi|fd] [<port>:]<data> [,<data>[...]]
        power
        port <port>
        temp
GR 66> quit
super>
```

Here are brief descriptions of **grrmb** commands; **reset**, **echo**, **help**, and **ver** are not available.

? - lists **grrmb** command set

bignore - displays broadcast ignore status, on or off

fan - displays chassis fan RPMs or on/off status

Output for a GRF 400:

```
GR 02> fan
Fan 0 : Curr 75 Last 9 Diff 66
Fan 1 : Curr 229 Last 162 Diff 67
Fan 2 : Curr 179 Last 112 Diff 67
```

Output for a GRF 1600, 1 = on, 2 = off:

```
GR 15> fan
Fan 0 : Curr 1 Last 0 Diff 1
Fan 1 : Curr 1 Last 0 Diff 1
```

maint - operates media-specific commands, described in media configuration chapters in *GRF Configuration Guide*

power - displays on/off status of GRF 1600 power supplies:

```
GR 15> power
power {1|2} {on|off}
```

port *number*	- sets GRF slot number, from 0–3 or 0–15
temp	- returns current temperatures measured at the control board

? or help
(CLI)

help and its alias **?** return a list of all registered commands authorized by user's security profile. Use **help** or **?** followed by a specific command name to obtain description or usage information.

Permission level: user

Syntax

help, **?**

help ***command-name***, **?** ***command-name***

Example

```
super> ?  whoami
whoami  - output the user profile name associated with this session
super>
```

aitmd
(CLI)

The **aitmd** program is a daemon process that is responsible for managing IP tunnels using the Ascend Tunnel Management Protocol (ATMP). **aitmd** can support a large number of ATMP tunnels to many home networks (HN). At present the GRF functions as a home agent operating in gateway mode. That is, traffic from a mobile node that has been tunneled to the GRF by a foreign agent (FA) is de-encapsulated and forwarded directly to the appropriate home network.

Messages are logged via **syslog** for major events such system start up and reconfiguration, and for operation errors such as failed negotiation attempts.

aitmd communicates with the frame relay daemon, **fred**, to get the status of gateway circuits to the home networks. **aitmd** command options are used primarily for debugging.

By default, **aitmd** is not running. To enable ATMP, the administrator creates the file `/etc/aitmd.run`. This is done in the UNIX shell with the command:

```
# touch /etc/aitmd.run
```

Save the configuration change with:

```
# grwrite
```

The daemon starts within 15 seconds and restarts automatically on future reboots. The file is removed normally with **rm** and the removal again saved with **grwrite**. Removing `/etc/aitmd.run` does not kill the daemon, it only prevents **aitmd** from being restarted.

The daemon loads the ATMP configuration from the file `/etc/aitmd.conf`. The configuration can be changed on the fly by editing the configuration file and then sending the process a HUP signal.

Syntax

aitmd [**-F**] [**-f *config_file***] [**-h**] [**-l *log_file***] [**-L *logopt***] [**-q**] [**-Q**] [**-d**] [**-P *pvc_fake_state***]

Options

-F

- sets **aitmd** to run in the foreground.

By default, the daemon disconnects from the controlling tty and forks itself into the background. The **-F** flag is useful during debugging and to prevent a shell wrapper from losing track of the daemon.

-f *config_file*]

- loads the configuration from the named file rather than the default `/etc/aitmd.conf`

-h

- help flag, prints a brief usage message

-l *log_file*

- sends log messages to the named file

This implicitly enables file logging in addition to the `syslog` logging.

-Q

- sets operation to quiet mode in which no log messages are generated

-P *pvc_fake_state*

This option tells the daemon to not check the real state of the ATMP circuits, and to assume that all circuits are in the given state. ***pvc_fake_state*** must be set to either up or down. This option is sometimes useful for testing configuration files for **aitmd** when the circuits are not actually up.

-L *logoption*

- sets a logging option

The arguments for a logging option are a pair of letters such as LN. The first letter is the name of the logging category or control that the option applies to, and the second is the option setting.

You can group logging options. For example, if you are debugging a complicated problem and want to run the daemon in the foreground, disable syslogging, enable logging to **stderr,** and see all logging categories, use this group of options (no spaces are needed between the grouped options `TsTfLA`):

```
# /usr/sbin/aitmd -F -L TsTFLA
```

The command and options can also be entered as:

```
# /usr/sbin/aitmd -F -L Ts -L TF -L LA
```

Logging can be targeted to go to **syslog**, or to a file, or both. If a file name is not specified, then "file" log output goes to **stderr**. The second letter is typically an on/off toggle, with upper case denoting on and lower case denoting off.

-L Ts	disable logging to syslog
-L TS	enable logging to syslog, default
-L Tf	disable logging to file or **stderr**, default
-L TF	enable logging to stderr or to a file, implied by -f, described above

Logging can be enabled and disabled by individual message category.

There are four categories of messages: notice, operational error, debug, and internal errors. The category can also be wildcarded to apply the control to all categories at once.

-L Ln	disable logging notice messages
-L LN	enable logging notice messages, default These messages are syslogged at syslog's LOG_NOTICE logging level and include non-error events such as start up and reconfiguration.
-L Lo	disable logging operational errors
-L LO	enable logging operation errors, default These errors are syslogged at syslog's LOG_WARNING logging level and are transient errors that can occur during normal operations, such as hosts being unreachable or authentication failures.
-L Ld	disable debug log messages, default
-L LD	enable debug log messages These are syslogged with syslog's at LOG_DEBUG logging level.
-L Le	disable internal error log messages
-L LE	enable internal error log messages, default These messages should never be seen during normal operations and usually indicate a bug. They are syslogged with syslog's at LOG_ERR logging level.

-L La	disable all log message categories
-L LA	enable all log message categories

Additional options are available to tailor what information is displayed in the log messages.

These controls can be applied to separately to each category, or to all categories at once. The first letter denotes the message category:

n for notices
o for operational errors
d for debug messages
e for internal errors
A to apply the control to all categories at once (wildcard)

The examples below all use the "**n**" (notice) category, but any category, or the wildcard **A**, could be used.

-L nt	do not include timestamp in notice log messages
-L nT	include timestamps in notices
-L np	do not include process IDs (pids) in notices
-L nP	include timestamps in notices

Signals

The tunnel management daemon will catch and handle a number of different signals. Signals are sent from the UNIX shell using the **kill** command.

HUP

This signal reloads the configuration file and reconfigures ATMP.

If a home network or foreign agent entry is deleted from the configuration file, then any tunnels associated with the agent or network are shut down. The status of the home network circuits is also rechecked. Tunnels that are negotiated before the reconfiguration and are valid in the new configuration continue to operate without interruption.

INFO

This signal causes the tunnel management daemon to log its current configured state. If a large number of tunnels are configured, this can take some time, and the daemon does not process new messages until this logging is complete.

TERM

This signal shuts down all tunnels and exits. Note that the tunnel manager is started by **grstart** and a "shell wrapper" exists to restart the daemon. Normally the daemon is automatically restarted after receiving this signal.

auth
(CLI)

This command selects and authenticates a User profile. Use this command to increase or decrease the permissions of the current login.

Permission level: user

Syntax

auth *user-name*

Example

If you supply the proper password for that User profile, the GRF enables the privileges in that profile and then displays the system prompt again. Note that the User profile may specify its own system prompt, which is a useful way to flag certain permissions levels; for example:

```
GRF> auth admin
Password:
admin>
```

If you supply the wrong password at the prompt, you'll see this message:

```
Login incorrect
User:
```

Enter the user name again and the Password prompt is displayed.

biosver
(CLI+shell)

The **biosver** command prints out the BIOS release date. The date is used to determine which BIOS version is installed on a GRF system. The command does not work on RMS

Permission level: system

Syntax

biosver

Example

```
super> biosver
Date BIOS was built: 04/17/97
```

bredit
(shell)

The **bredit** utility is used to access and edit the `bridged.conf` configuration file.

bredit opens the configuration file in the **vi** editor. After you make changes, you exit the file with the **vi** exit file **:q** command.

At this point **bredit** runs a script in which you are asked if you want to make the changes permanent. The script also gives you the option of signaling **bridged** to re-read the updated file immediately. When this option is taken, **bridged** restarts as if it was stopped and restarted for the first time. If you change the file in **vi** but do not choose either of the script options, **bredit** tells you that your changes were not committed.

If **bridged** is not running when **bredit** is used, the user is given the option of saving changes to the configuration in `bridged.conf` so that the next time **bridged** is started, the new changes take effect.

Syntax

bredit

brinfo
(shell)

The **brinfo** command is used to retrieve bridging interface information for administrative debugging and other situations where a simple checking of bridge port information is needed.

brinfo prints the list of bridge groups and the bridge ports (underlying interfaces) that are members of the specified group(s). If no groups are specified, all groups are reported on by default. **brinfo** gets its information directly from the BSD kernel whereas **brstat** gets its information from **bridged**.

Syntax

brinfo ***bridge_group*** **| all**

brstat
(shell)

The **brstat** command provides a snapshot of state information directly from **bridged**. A short lag occurs between the time a request is made and when an active **bridged** returns the information. **brinfo** gets its information directly from the BSD kernel whereas **brstat** gets its information from **bridged**.

Syntax

brstat

cd
(CLI)

Lists the field in the current profile. After a profile is read, you can view the contents of that profile by using the **cd** command (**cd** is an alias for the **list** command).

A second use is to move back (or up) in a profile. The **cd ..** command moves you back to the level you were viewing prior to the last **list** or **cd** entered.

See also: **list**, **get**, **ls**

Permission level: user

Syntax

cd [*field-name*] [***field-index***] [...]
cd .. [..]

Example

cd displays the fields in the User default profile after the profile is read:

```
super> read user default
USER/default read

super>cd
name* = default
password = ""
auth-method = { PASSWD { "" 1645 udp "" } { 5500 udp 5510 tcp +
active-enabled = yes
allow-system = no
allow-update = no
allow-password = no
allow-debug = no
prompt = GRF=>
log-display-level = none
```

clear
(CLI)

The **clear** command clears the screen and moves the prompt to the top line of the screen.

Permission level: user

Syntax

clear

csconfig
(shell)

csconfig is a UNIX command that sets PCMCIA slot interface on (up) or off (down), and reports general interface and device status. This command is useful for remote management of PCMCIA devices just to verify the status of device and slot interface readiness. If an external device fails or otherwise cannot be reached, error messages are sent to `grconslog.log`, they are not handled by **csconfig**.

Syntax

csconfig *slot_number* [up | down]

csconfig -a [up | down]

Options

-a	- apply command to all (both) slots
down	- set PCMCIA interface off, disable link to kernel
slot_number	- PCMCIA slots are A and B, numbers 0 and 1, respectively
up	- set PCMCIA interface on, enable link to kernel

Examples

Turn the interface in slot A (number 0) off before removing the PCMCIA card:

```
# csconfig 0 down
```

Use the commandto determine the status of the interface link to the kernel (up or down) and the state of the slot (empty or running). In this example, the interface can communicate with the kernel (up), but slot A is empty. When the interface is up, the slot is ready for a device to be inserted.

```
# csconfig 0
Slot 0: flags=0x5<UP,EMPTY>
```

In the next example, the interface is up and slot B contains an ATA flash device:

```
# csconfig 1
Slot 1: flags=0x3<UP,RUNNING>
        Attached device: wdc1
        Manufacturer Name: "SunDisk"
        Product Name: "SDP"
        Additional Info1: "5/3 0.6"
        Function ID: 4 (PC card ATA)
        Assigned IRQ: 11
        Assigned I/O port1: 0x3d0-0x3df
```

Use the **-a** option to return status for both slots, or to set both slots either up or down:

```
# csconfig -a
Slot 0: flags=0x5<UP,EMPTY>
Slot 1: flags=0x5<UP,EMPTY>
```

csctl
(shell)

The **csctl** is a UNIX utility for **csctld**, and is rarely needed by users. The **csctl** utility is used to configure and control PCCARD slots. If no arguments are specified, a list of possible PCMCIA device configurations are displayed.

Syntax

csctl [**-acdlqr**] [*conf_file*]

Options

-a	- Add a new device.
-c	- Check a configuration file.
conf_file	- If ***conf_file*** is not specified, `/etc/pccard.conf` is used, if required.
-d	- Run a daemon other than **csctld**.
-l	- Load in a new configuration.
-q	- Be quiet when the device is not configured.
-r	- Reset the configuration.

date
(CLI)

Returns current time and date.

Permission level: update

Syntax

date

Example

```
super> date
Tue Aug 12 19:02:35 1997
```

delete
(CLI)

The delete command is used to remove an instance of a profile from local storage. The specification of *`profile-index`* or *`profile-type`* is not optional. Only indexed profiles can be deleted. The Load, Dump, and System profiles cannot be deleted since only one of each exists.

Warning: After a deletion, the profile is gone permanently, there is no undelete or undo.

Permission level: update

Syntax

delete *profile-type profile-index*

Examples

You are always queried when you issue a **delete** command:

```
super> delete user operator
Delete profile USER/operator? [y/n] y
USER/operator deleted
```

If the profile does not exist, you get a message:

```
super> delete user operator
error: specified profile not found
super>
```

You can delete a field in a profile:

```
super> read card 1
CARD/1 read
super> delete port 1
Delete ports/1? [y/n] y
ports/1 deleted
```

dir
(CLI)

Returns a list of the top level or main-line profiles.

Permission level: user

Syntax

dir [***profile-type*** [***profile-index***]]

Example

To see the list of available profile types:

```
super> dir
CARD                  Card info
DUMP                  System dump information
LOAD                  System load information
SYSTEM                System info
USER                  Administrative user accounts
```

A **dir** request for a profile of a specific type returns the information for every known instance of that profile type.

In the output, the first column is the size of the profile in bytes, the second and third columns are the date and time the profile was last modified, and the fourth column lists the indices of the profiles.

```
super> dir user
92  10/09/96 11:19:31  default
103 10/09/96 11:19:31  admin
106 10/09/96 11:19:31  super
```

A **dir** request for a specific profile:

```
super> dir user default
92  10/09/96 11:19:31  default
```

exit
(CLI)

This command exits a user from the command-line interface (CLI), same as **quit**.

Syntax

exit

Permission level: user

fan
(grrmb)

This **grrmb** command displays the RPMs for GRF 400 chassis fans and fan on/off status for GRF 1600 chassis.

Syntax

fan

Examples

Here is output for a GRF 400:

```
GR 02> fan
Fan 0 : Curr 75 Last 9 Diff 66
Fan 1 : Curr 229 Last 162 Diff 67
Fan 2 : Curr 179 Last 112 Diff 67
```

You will see marked changes if you execute **fan** several times in a row. This is normal because the GRF 400 fans are constantly changing speed.

```
GR 1> fan
Fan 0 : Curr 127 Last 66 Diff 61
Fan 1 : Curr 206 Last 144 Diff 62
Fan 2 : Curr 194 Last 132 Diff 62
GR 1> fan
Fan 0 : Curr 176 Last 87 Diff 89
Fan 1 : Curr 7 Last 172 Diff 91
Fan 2 : Curr 254 Last 163 Diff 91
```

Here is what you will see if fan 0 has stopped:

```
GR 1> fan
Fan 0 : Curr 0 Last 0 Diff 0
Fan 1 : Curr 206 Last 144 Diff 62
Fan 2 : Curr 194 Last 132 Diff 62
```

Here is output for a GRF 1600, 0 is off, 1 is on:

```
GR 15> fan
Fan 0 : Current status 1
Fan 1 : Current status 1
```

fastboot
(CLI)

On RMS node-based systems, **fastboot** reboots the system. File systems are not checked at reboot time when you use **fastboot** and there is no command query. This allows the management software to load faster.

On GRF systems with the new control board, **fastboot** is the same as **reboot**.

Permission level: system

Syntax

fastboot [**-d -i -n -q**]

Options

-d	- if the -d option is specified, the system will make a ``crash dump" (debugging image) before halting or rebooting
-i	- the -i option is only available on the GRF system. This option is an override flag to ignore file system check. The GRF system has a built-in safety feature where it prevents the administrator from accidentally rebooting the system without saving the configuration files in `/etc`. It determines this by running the **grwrite** command. If the **grwrite** command exits with a non-zero exit status, then a warning message is printed and the shutdown operation is aborted. The -i option is used to override this safety feature. This option is not applicable on a non GRF system.
-n	- if the -n option is specified, the file system cache is not flushed. This option should probably not be used.
-q	- if the -q option is specified, the system is halted or restarted quickly and ungracefully, and only the flushing of the file system cache is performed. This option should probably not be used.

flashcmd
(CLI+shell)

The **flashcmd** command is used to execute a command against a specified flash device.

flashcmd mounts the flash device, performs the specified command against it, and then unmounts (**umountf**) the flash device. The default action is to use the device from which the system was loaded.

Permission level: system

Syntax

flashcmd [**-e**] [**-i**] [**-A**] [**-B**] [**-d** ***device_descriptor***] [**-w**] [**-r** ***revision***] ***command***

Options

-e	- use the device descriptor for the external flash device: `/dev/wd1a`
-i	- use the device descriptor for the internal flash device: `/dev/wd0a` (this is the default)
-A	- (option not available) use the device descriptor for the external flash device in PCMCIA slot A, `/dev/wd1a`
-B	- (option not available) use the device descriptor for the external flash device in PCMCIA slot B, `/dev/wd2a`
-d ***device_descriptor***	- use the specified device descriptor, this option enables another device to be acted upon
-w	- mount the flash device as writable The default is to mount the device as read-only.
-r ***revision***	- specifies the software release and version against which to perform the command This option is in the form *release,version*. For example, `1.4.x,default` or `1.4.x,atmtestB`. The currently-running version is assigned by **flashcmd** to be `default`.
command	- ***command*** can be any UNIX command. Ones typically used include copy (**cp**), determine free space (**df**), remove file (**rm**), and list contents (**ls**).

%R - substitutes the ***revision*** specified with the **-r** option in the ***command*** executed (see examples)

Examples

1 This command

```
# flashcmd -w rm -rf /flash/tmp
```

causes the following actions to be performed:

- mount the internal flash device
- remove the `/flash/tmp` directory
- unmount the device

2 The command:

```
# flashcmd -e -r 1.4.x,default cp /flash/etc.%R/grifconfig.conf
                                         /etc/grifconfig.conf
```

causes the following actions to be performed:

- mount the flash device contained in the external PCMCIA slot using the **mountf** command
- execute the **cp** command to copy the file from `/flash/etc.1.4.x,default/grifconfig.conf` on a flash device to the `/etc/grifconfig.conf` file on RAM

  Note that the `%R` "shorthand" translates to the specified release, `1.4.x,default`
- unmount the flash device using the **umountf** command

3 This command shows that %R represents the current system:

```
# flashcmd echo %R
1.4.x,default
```

4 These two commands perform the same actions:

```
# flashcmd ls /etc.%R
# flashcmd ls /etc/1.4.x,default
```

gdc
(CLI+shell)

Monitors and manages the **gated** routing daemon.

Permission level: system

Syntax

gdc [**-q**] [**-n**] [**-d**] [**-O** *size*] [**-c** *coresize*] [**-f** *filesize*] [**-m** *datasize*] [**-s** *stacksize*] [**-t** *seconds*] *command*

Options

-q

Run quietly. With this option informational messages which are normally printed to the standard output are suppressed and error messages are logged via **syslogd** instead of being printed to the standard error output.

-n

Run without changing the kernel forwarding table. Useful for testing, and when operating as a route server which does no forwarding.

-c ***coresize***

Sets the maximum size of a core dump a **gated** started with **gdc** will produce. Useful on systems where the default maximum core dump size is too small for **gated** to produce a full core dump on errors.

-d

-f ***filesize***

Sets the maximum file size a **gated** started with **gdc** will produce. Useful on systems where the default maximum file dump size is too small for **gated** to produce a full state dump when requested.

-m ***datasize***

Sets the maximum size of the data segment of a **gated** started with **gdc**. Useful on systems where the default data segment size is too small for **gated** to run.

-O ***size***

Used with **gdc start** to change the allowed number of open file descriptors.

This version of GateD removes a restriction on the number of open file descriptors which limited GateD to approximately 50 simultaneous adjacencies and/or BGP peering sessions. The default is now 320 open file descriptors, enough for roughly 300 adjacencies or peers.

-s ***stacksize***

Sets the maximum size of stack of a **gated** started with **gdc**. Useful on systems where the default maximum stack size is too small for **gated** to run.

-t ***seconds***

Specifies the time in seconds which gdc will spend waiting for gated to complete certain operations, in particular at termination and startup. By default this value is set to 10 sec.

Commands

The following commands cause signals to be delivered to **gated** for various purposes:

backout

Back out to the previous `/etc/gated.conf` file.

BACKOUT

Back out without questions.

checkconf

Check the current `/etc/gated.conf` file for syntax errors.

checknew

Check the new `/etc/gated.conf` file for syntax errors.

createconf

Create a new `/etc/gated.conf` file.

COREDUMP

Sends an abort signal to **gated**, causing it to terminate with a core dump.

dump

Signal **gated** to dump its current state into the file `/var/tmp/gated_dump`.

Caution: Note that a **gdc** dump will create a `/var/tmp/gated_dump` file under the uid of `nobody` and a gid of `none`. This prevents a `/var/tmp/gated_dump` file from ever filling the file system to more than 95% full.

Also note, it may be impossible on an Ascend GRF to edit (vi) the `/var/tmp/gated_dump` file due to file system size restriction of the RAM-based file system. It is recommended that `more | grep | ftp` to another machine is used to access or edit the file.

interface

Signal **gated** to recheck the interface configuration.

KILL

Cause **gated** to terminate ungracefully. Normally useful when the daemon has hung.

modeconf

Clean up `/etc/gated.conf` file modes.

newconf

Rotate the new `/etc/gated.conf` file into place.

reconfig

Signal **gated** to reread the `/etc/gated.conf` file configuration file, reconfiguring its current state as appropriate.

restart

Stop and then start **gated**.

rmcore

Remove existing **gated** core file.

rmdump

Remove existing **gated** dump file.

rmparse

Remove existing **gated** parse error file.

running

Determine whether **gated** is running.

start

Start **gated.**

stop

Signal **gated** until it stops.

term

Signal **gated** to terminate after shutting down all operating routing protocols gracefully. Executing this command a second time should cause gated to terminate even if some protocols have not yet fully shut down.

toggletrace

If **gated** is currently tracing to a file, cause tracing to be suspended and the trace file to be closed. If **gated** tracing is current suspended, cause the trace file to be reopened and tracing initiated. This is useful for moving trace files.

version

Display **gated** version information.

get
(CLI)

Displays the contents of the current working profile read into local memory. The **get** command retrieves the names and contents of fields within profiles without changing the user's location within the tree. Using **get**, you can look at a field in another profile. **ls** is an alias for **get**.

See also: **cd**, **list**, **ls**

Permission level: user

Syntax

get [. | ***profile-type*** [***profile-index***]] [***field-name field-index*** ...]

Example

Use get to check the contents of one profile when you are at a lower level of another profile:

```
super> read card 3
CARD/3 read
super> list .
card-num* = 3
media-type = no-media
debug-level = 0
hssi-frame-protocol = Frame-Relay
sonet-frame-protocol = PPP
ether-verbose = 0
ports = < { 0 {off on 10 3 }{single off}{"" "" 1 sonet internal-oscillat+
load = { 0 < > 1 0 0 }
dump = { 0 < > off off }config = { 0 0 1 1 4 0 }
icmp-throttling = { 0 10 10 2147483647 10 10 }

super> list ports 0
port_num = 0
cisco-hdlc = { off on 10 3 }
fddi = { single off }
sonet = { "" "" 1 sonet internal-oscillator 0 207 }
hssi = { 0 16-bit }
ether = { autonegotiate }
hippi = {1 32 no-mode 999999 4 incremental 5 300 10 10 03:00:0f:c0
                     disabled di+
super> list hssi
source-clock = 0
CRC-type = 16-bit
```

Here is where you use **get** to look in to another profile:

```
super> get card 1 ports 0 hssi
source-clock = 0
CRC-type = 32-bit
```

get operates on the last profile read when a dot (.) is used in place of the profile type and profile index:

```
super> read user default
USER/default read
super> get .
name* = default
password = ""
auth-method = {PASSWD {"" 1645 udp ""}{5500 udp 5510 tcp /var/ace}}
active-enabled = yes
allow-system = no
allow-update = no
allow-password = no
allow-debug = no
prompt = GRF=>
log-display-level = none
```

get operates on the last profile read when a dot (.) is used in place of the profile type and profile index and a field name is specified:

```
super> read user default
USER/default read

super> get . auth-method
auth-type = PASSWD
rad-auth-client = { "" 1645 udp "" }
securid-auth-client = { 5500 udp 5510 tcp /var/ace }
```

getver
(CLI+shell)

The **getver** command reports the release and version of the operating system that is currently running, or the release and version number of the system that is set to run after the next boot. Use the **setver** command to specify (set) the release and version of the next system to be run.

Permission level: system

See also: **setver**, **grfins**

Syntax

getver [**-c**] [**-n**] [**-s**]

Options

The **getver** command supports the following options:

-c	- reads the `/etc/release` file in system RAM and returns the release name and version of the currently running system (default)
-n	- reads the `/flash/etc/release` file in the internal flash drive and returns the release name and version of the system to be installed during the next boot
-s	- returns the release name and version in a binary or compact form for use in other scripts

Example

This pair of commands returns the names of the current and the “next-boot” operating systems:

```
# getver
Current Revision: 1.4.2.1  Version: default

# getver -n
Next Revision: 1.4.3.1  Version: fdditest

# getver -s
1.4.2.1,default
```

grarp
(CLI+shell)

grarp supports address resolution for all media. The command maintains a mapping of IP addresses to physical addresses in the `grarp.conf` configuration file and displays the file's contents. In addition to the information presented here, a man page is also available.

The `grarp.conf` file is a statically-configured global address resolution table. Each media card maintains its own address resolution table. The default behavior of the **grarp** command when displaying or deleting entries is to use the media card tables.

The **gria** command that mapped IP addresses to physical HIPPI card interfaces is deleted. It is replaced by **grarp**.

Note: Sites do not have to recreate **gria** entries. A built-in script automatically copies any existing **gria** command information into the site's new `grarp.conf` file.

Permission level: system

Syntax

grarp ***hostname*** [**-n**] [**-i** ***ifname*** | **-p** ***port***]

grarp **-a** [**bsd**] [**kmem**] [**-n**] [**-i** ***ifname*** | **-p** ***port***]

grarp **-d** ***hostname*** [**-i** ***ifname*** | **-p** ***port***]

grarp **-s** ***hostname*** ***phys_addr***
[**-e**] [**temp**] [**pub**] [**trail**] [**-i** ***ifname*** | **-p** ***port***]

grarp **-f** ***filename*** [**-i** ***ifname*** | **-p** ***port***]

grarp [**-i** ***ifname*** | **-p** ***port***]

Arguments

hostname	- specifies the host by name or by address (using Internet dot notation)
ifname	- the GRF interface name in the format `gx0yz`
phys_addr	- the format of ***phys_addr*** used with the **-s** option depends on the type of media: **FDDI** - 6 hexadecimal bytes separated by colons **HIPPI** - a 32-bit unsigned integer in C language convention (prefix "0x" for hexadecimal, "0" for octal, a non-zero digit for decimal) **ATM** options - 1: specifies address and an optional subaddress as a single string with "+" separating them (address and subaddress are

	each a series of up to 20 hexadecimal bytes, two digits each, with optional "." separators between the bytes) 2: uses VPI/VCI values for a PVC in the form `nn/mm` where `nn` is VPI and `mm` is VCI) 3: with **-e**, address is in CCITT E.164 format
port	- the slot number or a GRIT address of the form `0:<slot>:<interface>`

Command options

-a	- displays all current ARP entries by reading tables from the file `kmem` (the default is `/dev/kmem`) based on the kernel file `bsd` (default `/bsd`), and from the media cards
-d	- deletes an entry for the host called ***hostname*** (superuser only)
-e	- (ATM only) specifies that the ATM address is in CCITT E.164-style instead of the default NSAP format (if an ATM subaddress is given, this option is overridden and the address and subaddress are treated as E.164 and NSAP, respectively)
-f	- causes ***filename*** to be read and multiple entries to be set in the ARP tables Entries in ***filename*** should be of the form: ***ifname hostname phys_addr*** [**temp**] [**pub**] [**trail**]
-i *ifname*	- restricts command actions to BSD kernel
-n	- shows network addresses as numbers (normally **grarp** interprets addresses and attempts to display them symbolically)
-p *port*	- restricts command actions to a media card
-s *hostname phys_addr*	- creates an ARP entry for the host called ***hostname*** with the physical address ***phys_addr***

Other options

pub	- “publishes” entry so that this system acts as an ARP server and responds to requests for ***hostname*** even though the host address is not its own
temp	- specifies entry is temporary
trail	- enables trailer encapsulations to be sent to ***hostname***

grass
(CLI+shell)

The **grass** command is used to enable and disable various internal IP service settings in the operating system. The `/etc/grass.conf` file should not be edited by the user.

grass allows an administrator to choose what application level services are run, and to quickly enable/disable them as needed. If possible, **grass** attempts to activate/deactivate the service immediately. The user is informed of the actions taken by the program. Changes made using **grass** must be saved using **grwrite** to save the changes for the next reboot.

By default, **grass** displays the current status of all managed services as viewed by their configuration files. Services are listed in the `/etc/services` file.

Permission level: system

Syntax

grass [**+/-s** ***service_name***] [**+/-g** ***group_name***] [**+/-1**]

Options

+/-s ***service_name***	- activates/deactivates the named IP service The service is listed in the `/etc/grass.conf` configuration file, and disabled or enabled as required. This option can be repeated to add or subtract multiple services from the set of services to change.
+/-g ***group_name***	- activates/deactivates a group of IP services The group name is listed in the `/etc/grass.conf` file and each service in the group is enabled/disabled. This option can be repeated to add or subtract multiple groups of services from the set of services to change.
+/-l	- lists the services and groups from `/etc/grass.conf` For each service, information is displayed: - name of the service - file in which service entry is located - script run to change service status "on the fly" - the regular expression string used to locate the service in the given file Group entries contain the list of services managed by the group.

Examples

Line options are cumulative. For example, this command disables all services except `telnet:`

```
# grass -g all +s telnet
```

This command disables `telnet,` and prevents remote access to the system:

```
# grass -s telnet
```

Enacting changes

If a script to dynamically start/stop the given service is not available, the user will need to reboot the system to enact changes.

etc/services

The `etc/services` file maps the protocol names to the TCP/IP ports.

When you modify the `/etc/services` file, those changes will get saved to flash memory when you do a **grwrite**. On RMS node systems, your changes to `/etc/services` are saved and archived by the **grc** command. However, subsequent software upgrades will install the new release version of `/etc/services`, overwriting any changes you may have made. Please be sure to record your changes to this file as you will need to add them again when you upgrade.

Here is a portion of the `/etc/services` file:

```
# Network services, Internet style
#       @(#)services    8.1 (Berkeley) 6/9/93
gritlog         16/grit
gritboot        17/grit
gritpclog       18/grit
grinch          19/grit
echo            20/grit
discard         21/grit       sink null
daytime         22/grit
chargen         23/grit       ttytst source
time            24/grit
trap            25/grit
ioctl           26/grit       BSD kernel ioctl routines
filtercomm      27/grit       Filterd known port
#
tcpmux          1/tcp                # TCP port multiplexer (RFC1078)
echo            7/tcp
echo            7/udp
discard         9/tcp         sink null
discard         9/udp         sink null
systat          11/tcp        users
daytime         13/tcp
daytime        13/udp
netstat         15/tcp
chargen         19/tcp        ttytst source
chargen         19/udp        ttytst source
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp        mail
time            37/tcp        timserver
time            37/udp        timserver
rlp             39/udp        resource      # resource location
nameserver      42/tcp        name          # IEN 116
whois           43/tcp        nicname
domain          53/tcp        nameserver    # name-domain server
domain          53/udp        nameserver
```

gratm
(CLI+shell)

The **gratm** program configures ATM OC-3c and OC-12c media cards with specific information needed for ATM operations. This includes information for configuring:

- physical interfaces for SVC signalling
- permanent virtual circuits (PVCs)
- rate queues to support traffic shaping

gratm reads configuration information from `/etc/gratm.conf` and uses the **grinch** command to download the configuration information to the appropriate media card. Please see the *GRF Configuration Guide* for a template of the `gratm.conf` file.

Permission level: system

Syntax

gratm [**-n**] [**-f** ***file***] ***card*** [**...**]

file	- name of the new program file
card	- the first four components of the interface name, `ga0yz`, minus the `z` component, (the logical interface number) – identifies a specific ATM media card as to slot location

Options

-f *file*	- reads the ATM physical interface configuration from the specified file instead of the default configuration file `/etc/gratm.conf`
-n	- no execute: prints the **grinch** commands that would be used to configure the kernel or media card, but does not actually execute them

Examples

The **gratm** command can be used to check the syntax of the `gratm.conf` configuration file before the configuration is used to initialize ATM media cards.

To parse and check the syntax of `/etc/gratm.conf`, enter:

```
# gratm
```

To parse and check the syntax of `/etc/gratm.conf` and print the **grinch** commands that would be run by **gratm** to configure the ATM card in slot 3 without actually performing any configuration actions, enter:

```
# gratm -n ga03
```

grbist
(CLI+shell)

The **grbist** command establishes a media card connection over which to run field-run diagnostics. Use only with direction from Technical Support.

Permission level: system

Syntax

grbist [**-p** ***portcard***]

grburn

This command is now obsolete, refer to the **grflash** command.

grc
(shell)

The **grc** script provides commands to collect, archive, and restore system configuration files on systems with RMS nodes. GRF systems without RMS nodes use **grsnapshot**.

See also: **grwrite**, **grsite**, **grsnapshot**

Syntax

grc [**info**] [**list**] [**restore**] [**save**] [**-a** ***archive***] [**-d** ***directory***] [**-f** ***listfile***] [**-F**] [**-q**] [**-v**]

Usages

These are the options appropriate for each **grc** command:

grc info [**-q**] [**-a** ***archive***] [**-d** ***directory***] [**-f** ***file***]

grc save [**-Fqv**] [**-a** ***archive***] [**-d** ***directory***] [**-f** ***file***]

grc restore [**-qv**] [**-a** ***archive***] [**-d** ***directory***] [**-f** ***file***]

grc list [**-qv**] [**-a** ***archive***] [**-d** ***directory***] [**-f** ***file***]

Commands

info	- prints information about the system configuration archive This includes the date and time the archive was created, the user who created the archive, and any comments supplied at creation.
list	- lists the configuration files in the archive
restore	- extracts all configuration files from the archive The files are placed relative to the current directory, or to the directory specified by the **-d** option.
save	- saves the configuration files in an archive Prompts the user for an optional comment to be stored in the archive and used to describe the contents. You might add the comment, “Config before adding new FDDI card to slot 3".

Options

Options are specified after the appropriate **grc** command. **grc** responds with an appropriate usage message if an inappropriate option is used with a command.

-a ***archive***	- writes information *to* (for the **save** command) or reads information *from* (for the **restore** command) the specified

archive instead of the default floppy diskette drive of `/dev/fd0`

-d ***directory*** - specifies the directory relative to which **grc** will save or restore the configuration files

This option has **grc** do a **cd** to *directory* prior to the operation.

-F - used with the **save** command to format the floppy diskette that will hold the archive (uses the **fdformat** command)

-f *listfile* - *r*eads the list of files to be archived, extracted or listed from *listfile*

Files are specified relative to the current working directory or the directory specified by the **-d** option, and are listed one per line.
A line in a *listfile* that begins with a `#` (pound sign) is considered a comment and is ignored.

-q - quiet mode, suppresses the usual chatty messages that describe what **grc** is doing

-v - verbose mode

When used with **restore** or **save** commands, the option lists files that are being extracted from, or written to, the archive.

When used with the **list** command, the option prints out a verbose listing (uses **-v** option of **tar**).

The default list of configuration files not only includes router-specific files, such as `/etc/grifconfig.conf`, but also includes typical configuration files that are subject to modification from time to time, such as `/etc/passwd`.

Note: Any changes you make to the `/etc/services` file are overwritten when you install a new software release. Record these changes and add them back after the upgrade.

```
etc/aitmd.conf           etc/gritd.conf         etc/namedb
etc/bridged.conf         etc/grlamap.conf       etc/netstart
etc/crontab              etc/group              etc/networks
etc/fstab                etc/grppp.conf         etc/passwd
etc/filterd.conf         etc/grpvc.conf         etc/printcap
etc/gated.conf           etc/grroute.conf       etc/pwd.db
etc/gettytab             etc/grstart            etc/rc
etc/grarp.conf           etc/hostname.txt       etc/rc.local
etc/gratm.conf           etc/hosts              etc/resolv.conf
etc/grclean.conf         etc/hosts.equiv        etc/services
etc/grclean.logs.conf    etc/localtime          etc/snmpd.conf
etc/grfr.conf            etc/master.passwd      etc/spwd.db
etc/grifconfig.conf      etc/motd               etc/syslog.conf
etc/grinchd.conf         etc/named.boot
```

Figure 1-6. List of files archived by the grc command

A site can specify alternate file(s) to archive on the command line, or in a *listfil*e specified by the **-f** option. If alternate files are specified both in the **-f** option file and on the command line, the **grc** script archives any file specified in either *listfil*e or the command, that is, **grc** will operate on a union of the two lists.

All files are specified relative to the directory from which the archive will be created (usually /). For example, the password file is specified `etc/passwd`.

By default, **grc** operates on a floppy diskette in the RMS node's floppy diskette drive (`/dev/fd0`). An alternate archive (e.g., a file) can be specified using the **-a** option. A man page is available.

You must log in as `root` to use **grc**.

grcard
(CLI+shell)

Use the **grcard** command to display the status of media cards, including slot number, hardware type, and current operating status.

Permission level: system

Syntax

grcard [**-dv**]

Options

-d	- specifies the debug option (not available)
-v	- specifies the verbose option to add column headers to command output

States

A media card will be in one of these operating states:

Power-up	- initial state of a card at system power on
Boot-requested	- card has requested its run-time code
Dumping	- card is being dumped
Loading	- card is receiving run-time code
Configuring	- card has requested its configuration tables
Running	- card is configured and operating
Not-responding	- card does not respond to requests from management software, or the media card is not installed in the slot
Panic	- card has encountered a system fault
Hold-reset	- card is being held in reset state
State unknown	- card's state cannot be determined

The "Q" type of media card can be indicated in a display. For example, FDDI/Q and ATM/Q cards appear as FDDI_V2, and ATM_OC3_V2, respectively. V2 does not indicate /Q versions, as HSSI_V1, SONET_V1, and ETHER_V1 are all /Q level versions.

As shown in this output from a GRF 400, the information displayed includes slot number, hardware type, and current operational status:

```
#  grcard -v
Slot    HWtype  State
----    ------  -----
0       HSSI_V1 running
1       HIPPI_V1 not-responding
2       ATM_OC3_V2      running
3       FDDI_V2 running
```

These are the names **grcard** displays for other cards: `ETHER_V1, SONET_V1, DEV1_V1, and ATM_OC12_V1.`

grclean
(CLI+shell)

grclean is an internal program that compresses, archives, and manages dumps and log files, and saves them to a specified file name.

Software release 1.3.11 changed the contents of the `/etc/grclean.logs.conf` file.

If you are upgrading to 1.4 from a 1.3.9 or earlier software release, the previous `grclean.logs.conf` file is renamed to `grclean.logs.conf.old,` and a new copy of `grclean.logs.conf` is installed. You may see the message describing this transfer if you are watching the console as **grfins** operates

If you have never made any changes to `grclean.logs.conf`, the upgrade has no effect. However, if you did change `grclean.logs.conf` in the past, then you must now make those changes again as soon as possible after the 1.4 update procedure is finished.

Cut and paste just your changes into the 1.4 version of the file, take care not to overwrite the new sections in the file.

Permission level: system

Syntax

/usr/nbin/grclean [**-f** ***configfile***] [**-l** ***log***] **-n** **-v**

Options

-f *configfile*	- dump to ***configfile*** instead of default, `/etc/grclean.conf`
-l *log*]	- log **grclean** events to ***log*** instead of default file, `/var/log/grclean.log`
-n	- do not execute, just show actions
-v	- display in verbose mode

grconslog
(CLI+shell)

The **grconslog** command displays the messages logged to `/var/log/gr.console` file. Various options exist to filter the messages displayed. If no options are specified on the command line, **grconslog** prints the message associated with every entry in `/var/log/gr.console`.

Common usage is **grconslog -vf**. Enter a <CtrlC> to quit the log.

Note: **grconslog** sometimes hangs when the log file fills up. Do a <CtrlC> and then restart **grconslog**.

The **grconslog** command runs normally when logging is configured to an external PCMCIA flash device or to an NFS-mounted disk. The `/var/log` directory has a pointer to the external device. The *GRF Configuration Guide* describes how to configure these options.

When logging is done remotely via **syslogd**, items are logged out to the **syslog** server. Note that **grconslog** runs locally. You must run **grconslog** on the remote **syslog** server or use the **tail -v** .

Permission level: system

Syntax

grconslog [**-** ***number***] [**-a**] [**-A**] [-{**b**|**B**} ***slot***] [**-d**] [**-e**] [**-f**] [**-h**] [**-p**] [-{**r**|**R**} ***regex***] [**-t**] [**-T**] [**-v**] [**-V**] [***file***]

Options

- ***number***	- print only the last ***number*** of lines in `/var/log/gr.console` The default is 10 lines. (Refer to the **tail** man page.)
-a	- print all lines, even blank lines and screen prompts
-A	- print all lines, including blank lines, screen prompts, date stamp, sending media card, and media card hardware type (effectively equivalent to **grconslog -atv**)
-b ***slot***	- only print the messages for media card in specified ***slot***.
-B ***slot***	- only print the messages for media cards *not* in specified ***slot***.
-d	- print the date stamp
-e	- expand

When a message is put in `/var/log/gr.console` to indicate the last message has been repeated *n* times, the last message is actually printed out *n* times.

Without **-e**, "*last message repeated* n *times*" is printed.

-f	- causes **grconslog** not to stop when end of file is reached, but to wait for additional data to be appended to the input (Refer to the **tail** man page.)
-h	- help, prints out all options, then exits
-p	- print slot number of message originator
-r ***regex***	- print all messages that contain a specified regular expression (***regex***) (Refer to the **perl** and **grep** man pages.)
-R ***regex***	- print messages that do not contain a specified regular expression (***regex***) (Refer to the **perl** and **grep** man pages.)
-t	- print the hardware type of the message originator
-T	- print the message type
-v	- verbose mode: print the slot number of message originator and print the date stamp (effectively equivalent to **grconslog -dp**)
-V	- version: print version number of **grconslog** and print `/etc/motd` to return current version level of system software
file	- instead of filtering `/var/log/gr.console`, **grconslog** filters ***file*:** If ***file*** is '-', then it filters standard input

grdebug
(CLI+shell)

This command controls operating system actions in response to the state of a media card. By default, the **grdebug** command prints the current configuration of the actions for the media card specified via the **-p** option.

Use these options with care, media card performance can be adversely affected.

Permission level: system

Syntax

grdebug [**-C**] [**-E**] [**-H**] [**-I**] [**-P**] [**-R**] [**-T**] [**-U**] **-p** ***slot number*** [**on**|**off**]

Options

slot number	- chassis slot number, from 0 to 3 (or 0 to 15)
-C	- specifies that the "*reset upon configuration errors*" action will be configured on, configured off, or displayed
-E	- specifies that the "*echo (ping) when silent*" action will be configured on, configured off, or displayed
-H	- specifies that the "*hold card in reset upon configuration errors*" action will be configured on, off, or displayed
-I	- specifies that the "*initialize card upon start up*" action will be configured on, configured off, or displayed
-P	- specifies that the "*reset card upon panic*" action will be configured on, configured off, or displayed
-p *slot number*	- specifies that the bits for the indicated media card will be configured on, configured off, or displayed
-R	- specifies that the "*reset card when hung*" action will be configured on, configured off, or displayed
-T	- specifies that the "*time out card for echo or reset*" action will be configured on, configured off, or displayed
-U	- specifies that the "*update card during normal operation*" action will be configured on, configured off, or displayed

Examples

To turn off all monitoring activity for a media card in slot 2, enter:

```
# grdebug -p 2 off
```

To prevent a media card in slot 2 from automatically having its configuration updated and from being reset when it does not respond to a **ping** command, enter:

```
# grdebug -R -U -p 2 off
```

gredit
(CLI)

Opens `filterd.conf`, `gated.conf` configuration files, but brings it up in **vi** to edit.

Permission level: system

Syntax

gredit filterd

gredit gated

grfddi
(CLI+shell)

This utility command sets the dual and single attachment connections for an FDDI media card. **grfddi** commands are not maintained across system reboots. Permanent settings must be made in the appropriate Card profile. After configuring or reconfiguring the attachment interface, you must reset the media card for the change to take effect.

Permission level: system

Syntax

grfddi [**-p** ***slot number***] [**-u** | **-l**] [**-s** | **-d**]

Options

slot number	- chassis slot number, from 0 to 3 (or 0 to 15)
-u	- specifies upper pair of interfaces, A0 and B0
-l	- specifies lower pair of interfaces, A1 and B1
-s	- sets specified interface pair as two single attach
-d	- sets specified interface pair as one double attach

Example 1

To set the lower pair of interfaces on FDDI media card in slot 3 to dual attach and set the upper pair to two single attach interfaces, first enter:

```
# grfddi -p3 -l -d
Configuring lower ports on slot 3 to be dual-attached.
2.12.2.4.4.3.4.1:  OK
2.12.2.4.4.4.4.1:  OK
Port card in slot 3 must be reset for change to take effect.
```

Then enter:

```
# grfddi -p 3 -u -s
Configuring upper ports on slot 3 to be single-attached.
2.12.2.4.4.1.4.1:  OK
2.12.2.4.4.2.4.1:  OK
Port card in slot 3 must be reset for change to take effect.
```

Example 2

Two commands are needed to set all interfaces on FDDI media card in slot 1 as single attach interfaces:

```
# grfddi -p 1  -l  -s
# grfddi -p 1  -u  -s
```

grfins
(CLI+shell)

The **grfins** command installs a given release of software on non-RMS node systems and up/downgrades the configuration files as required.

Use **grwrite** to save any configuration changes made since last boot.

```
# grwrite
```

The **grfins** command unpacks zipped files, **flashcmd** cleans up the flash device:

```
# grfins --source=/flash/tmp --release=1.4.1 --activate
# flashcmd -w rm -rf /flash/tmp
```

A reboot installs the files on system RAM, enter:

```
# shutdown -r now
```

or use:

```
# reboot -i
```

Permission level: system

Syntax

grfins --source=*source_loc* --release=*release_name* *options*

--source=*source_loc*	- specifies the source of files to be installed, required, there is no default
--release=*release_name*	- provides the release name as seen in the new release binary, must specify full release name, for example, `1.4.x`

Options

--activate	- executes the **setver** command when finished to make this installation activate on the next boot
--help	- shows a short help message
--move	- when placing the `.TAR.gz` and `root.gz` files, moves them instead of using a copy command
--no	- answers no to all queries One query is whether configuration information should be saved in `/etc/profs`. With the **--no** option, the information will not be saved in that directory.
--savesite	- does not delete the site file for this release, this is the default if the user is installing over an existing release
--srcflash	- selects an alternate source flash device If option is used, the device is mounted on `/mnt`. Thus, `--source=` value should start with `/mnt/` as a path to the source `.gz` files.

--target=*device_name*	- specifies the target device on which release files will be loaded, current device names are **internal** and **external**
--verbose	- prints out more event information during disk creation
--yes	- answers yes to all queries One query is whether configuration information should be saved in `/etc/profs`. With the **--yes** option specified, information is saved in that directory.

grflash
(CLI+shell)

The **grflash** command burns a new program into flash memory on all media cards. (This command replaces **grburn**.)

Warning: The **grflash** command must only be used with the direction of Customer Support staff.

Permission level: system

Syntax

grflash **-p** ***port_num*** [**-dh**] [**-v**] [**-E**] [**-T** ***timeout***] ***file***
-t { **a** | **aq** | **a2** | **e** | **f** | **h** | **hl** | **r** | **rl** | **r2** | **rx** | **s** | **d** }

port_num	- chassis slot number, from 0 to 3 (or 0 to 15)
file	- location of new program file

Options

-p ***port_num***	- directs **file** to a specified port
-d	- specifies debug option (not available)
-h	- turns on hash marks so they are displayed after every segment is sent
-v	- specifies verbose option, displays number of segments sent so far and total number of segments to be sent
-E	- places card in a series of states, checks for error conditions
-T ***timeout***	- specifies the number of seconds to wait for a packet from the card or control board, default is 60 seconds

-t { **a** | **aq** | **a2** | **e** | **f** | **h** | **hl** | **r** | **rl** | **r2** | **rx** | **s** | **d** } - specifies type of file being sent:

aq	ATM/Q boot loader
a2	ATM/OC-12 boot loader
e	Ethernet runtime program
f	FDDI runtime program
h	HIPPI runtime program
hl	HIPPI loader
o	SONET runtime program
r	RMB runtime program
rl	RMB loader
r2	RMB secondary loader
rx	RMB xilinx
s	HSSI runtime program
d	DEV1 runtime program

grfr
(shell)

The **grfr** program configures and manages Frame Relay parameters on GRF media cards.

The **grfr** command can be executed at any time to add, modify, or delete PVC configurations as well as to display various types of status.

The default configuration file used by **grfr** and by **fred**, the frame relay daemon, is `/etc/grfr.conf`.

Syntax

grfr -c ***command*** [**-n** ***pvc-name***] [**-s** ***slot***] [**-l** ***port***] [**-i** ***dlci***] [**-d** ***debug-level***]
[**-f** ***grfr-file***] [**-g** ***grif-file***]

Options

-c ***command***

- the **-c** flag is required when using a **grfr** command, commands begin on the next page

-n ***pvc-name***

- the name of a symbolic link or PVC, names are assigned in the `/etc/grfr.conf` file, for example, `name=agent1`

-s ***slot***

- specifies the number of the GRF chassis slot in which the card resides

-l ***port***

- specifies the port number, also called the link number

-i ***dlci***

- specifies the DLCI or channel identifier

-d ***debug-level***

- specifies the number of the debug level, range is 0–4

0	=	lowest level
1	=	useful when bringing up system, default
2	=	displays packets received and sent
3	=	displays packets received and sent, packet content
4	=	returns protocol-level information

-f ***grfr-file***

- retrieve Frame Relay configuration information from the named file instead of from the default `/etc/grfr.conf` file

-g ***grif-file***

- retrieve interface configuration information from the named file instead of from the default `/etc/grifconfig.conf` file

Display commands

Display commands are prefaced with the **-c** flag and begin with the letter **d**:

-c dsc
- display system configuration and status

-c dlc
- display link configuration and status

-c dpc
- display PVC configuration and status

-c dic
- display interface configuration and status

-c dss
- display system status

-c dls
- display link status

-c dps
- display PVC statistics

-c dbs
- display board status

Configuration commands

The **grfr** configuration commands are prefaced with the **-c** flag and start with the letter **c**.

In commands that enable a link or PVC, the second letter is **e.** In commands that disable a link or PVC, the second letter is **d**.

Configuration commands require links or PVCs to be specified. Use one or more of these options: **-i** ***DLCI***, **-l** ***port***, **-s** ***slot***.. (The **-n** ***name*** option is not available.)

-c cel
- enable link, enable link on slot 3, port 1: `# grfr -c cel -s 3 -l 1`

-c cdl
- disable link, disable link on slot 3, port 0: `# grfr -c cdl -s 3 -l 0`

-c cep
- enable PVC, requires PVC to be specified: `# grfr -c cep -i 888 -s 3 -l 0`

-c cdp
- disable PVC, requires PVC to be specified: `# grfr -c cdp -i 888 -s 3 -l 0`

grfr can also be used to configure new PVCs that have been added to the configuration file or disable PVCs currently in the file.

-c ccp

- configure PVC, requires the configuration file and the PVC to be specified:
`# grfr -c ccp -f /etc/grfr.conf -i dlci`

-c crp

- remove PVC, requires the PVC to be specified: `# grfr -c crp -i` *`dlci`*

The debug level can be displayed and set with **grfr**:

-c ddl

- display debug level: `# grfr -c ddl`

-c csd

- set debug level, requires the **-d** option to specify the new debug level
Debug levels are 0–4: `# grfr -c csd -d 3`

Examples

This command displays the statistics for configured Frame Relay PVCs. Enter:
`# grfr -c dps`

```
C O N F I G U R E D   P V C s   S T A T s:
=========================================
(S=Slot, P=Port, R=receive, T=Transmit)
(TP=Transmitted Packets, TO=Transmitted Octets)
Name       S/P/DLCI   Type    R-Packets  R-Octets  T-Packets T-Octets  TP-Dropped TO-Dropped
----       --------   ----    ---------  --------  --------- --------  ---------- ----------
2:0:0      02:0:0     Switch  2793       81044       2791     39074      0          0
2:0:160    02:0:160   ATMP    10266      1038795     10225    573187     0          0
2:0:491    02:0:491   Route   10270      863275      10279    1327565    0          0
```

This command shows the status of configured links and their current parameters. Enter:
`# grfr -c dlc`

```
    C O N F I G U R E D   L I N K S :
    =================================
    Name:         S/P:  LMI:     Link:    Autogrif: N391: N392: N393: T391: T392: Status:
    ----          ---   ---      ----     --------  ----  ----  ----  ----  ----  ------
    2:0           2 /0  ANNEX-D  UNI-DTE  None      6     3     4     10    15    Active
    Total: 1 links configured
```

grifconfig (shell)

The **grifconfig** command initializes and manipulates the configuration of GRF network interfaces. By default, **grifconfig** reads a table of interfaces from the file `/etc/grifconfig.conf` and compares these entries to those found in the kernel. Addresses that are no longer defined for an interface are removed and new ones are added. An interface is not modified unless an IP adddress, netmask, and (optionally) a broadcast or destination address has been modified.

Syntax

grifconfig [**-InR**] [**-f** ***file***] [***interface*** [...]]

grifconfig -c [**-InR**] ***interface ifaddress*** [***netmask***] [***address***] [***arguments*** [...]]

interface	- the GRF interface name in the format `gx0yz`.
ifaddress	- (optional) the Internet address of the interface. Use a single hyphen as a place-holder if no address is desired.
netmask	- (optional) the netmask for the interface. Use a single hyphen as a place-holder if no netmask is desired, but a broadcast or destination address must be specified in the next field.
address	- (optional) For interfaces on broadcast media (such as FDDI), the broadcast address for the interface. For interfaces on non-broadcast network media (such as ATM and HIPPI), the destination address of a point-to-point interface. If address is not specified or is the place-holder argument - (a single hyphen) on an entry for a non-broadcast interface, the interface will be treated as a multi-access interface (it is attached to an ATM or HIPPI switch rather than a single host).
arguments	- (optional) Additional arguments can be used to specify an MTU value or other descriptor for the interface. See the **ifconfig** man page for other argument options.

Options

-c	- specifies that instead of reading its arguments from the configuration file, **grifconfig** configures a single interface from the command-line arguments.
-f *file*	- requests that the table of interfaces will be read from the specified file instead of the default configuration file `/etc/grifconfig.conf`. A file name of - specifies that the table of interfaces will be read from standard input.
-I	- This option is obsolete and remains only for historical purposes.

-n	- specifies no execute option so that commands which would be used are printed but not actually executed.
-R	- instructs **grifconfig** to call the **grroute** command to initialize the kernel with the remote routes that are accessible via gateways on the network attached to the interface being configured.
-I	- specifies that the value command-line arguments (following the options) will specify the values of individual bytes in the packet, not entire words. The arguments for each byte must be presented in network byte order.

grinch
(CLI+shell)

The **grinch** command sends one or more copies of a GRINCH configuration request to a specified media card.

grinch interprets its command-line arguments as a list of values that specify the contents of each four-octet word of the packet data. Values are specified in standard C notation: a leading '0x' indicates a hexadecimal number; a leading '0' indicates an octal number; everything else is a decimal number. By default, **grinch** will send the packet to the media card specified by the **-p** option, or to the media card specified by the value of the GRID_PORT environment variable if the **-p** option is not used.

Permission level: system

Syntax

grinch [**-A**] [**-c** ***count***] [**-D**] [**-d**] [**-f** ***file***] [**-h** ***hwtype***] [**-i** ***interval***]
[**-p** ***slot number***] [**-q**] [**-v**] [**-x**] ***value*** [**. . .**]

Options

-B	- specifies that the value command-line arguments (following the options) will specify the values of individual bytes in the packet, not entire words The arguments for each byte must be presented in network byte order.
-c ***count***	- specifies that *count* copies of the packet will be sent to the destination media card By default, the packets will be sent one second apart.
-d	- specifies the debug option (not currently implemented)
-h ***hwtype***	- specifies that the value of the hardware type field in the GRID header should be set to the value for ***hwtype***
-i ***interval***	- specifies each copy of packet is sent interval seconds apart
-p ***slot number***	- specifies that the packet or group of packets will be sent to the destination media card in ***slot number***
-q	- specifies the quiet option in which **grinch** does not print any information about received packets
-v	- sets verbose option in which **grinch** prints contents of any unexpected packets received (usually ignored)
-x	- specifies that all integer variables will be printed in hexadecimal format (with a leading 0x)

grinstall
(shell)

The **grinstall** command installs new system software only on a GRF system that uses an RMS node. A site typically uses this command when it has downloaded a new version of software via ftp.

Unless otherwise specified, **grinstall** searches the ***directory*** `/usr/netstar/Release` for an archive of software for the indicated ***release***. It unpacks and installs the software contained in the archive. After a successful installation, the script writes a confirmation to the file `/usr/netstar/release/log`.

Refer also to the **grinstall** man page.

Syntax

grinstall ***release_name*** [**-nqtv**] [**-d** ***directory***] [**-f** ***file***] [**-s** ***script***]

Options

release_name	- name of GRF software release, 1.4.3, for example
-n	- suppresses log record of the installation
-q	- suppresses installation status messages from **grinstall**
-t	- prints a list of the files that will be extracted (equivalent to the **tar(1) -v** option)
-v	- when used with **-t** option, prints a verbose list of the files that will be extracted (equivalent to the **tar(1) -v** option)
-d ***directory***	- specifies the directory in which the **grinstall** script searches for the archived software. (if a file named.log exists in ***directory*** and the ***n*** or ***t*** options are not used, a record of the installation is logged to the file)
-f ***file***	- file from which **grinstall** extracts software instead of default ***release***
-s ***script***	- specifies that **grinstall** extract the specified ***script*** from the archive and use it to unpack the software instead of the default script **EXTRACT**

grlamap
(CLI+shell)

The **grlamap** script is used to initialize and manipulate the configuration of logical addresses on HIPPI media cards. When a HIPPI media card boots, **grinchd** uses **grlamap** to configure the logical addresses for that card.

By default, **grlamap** reads logical address configuration information from the file `/etc/grlamap.conf` and uses **grinch** to download the configuration information to the appropriate media card.

Permission level: system

Syntax

grlamap **-p** ***slot number*** [**-bdnq**] [**-f** ***config_filename***] [**-i** ***timeout_interval***]
[**-o** ***output_filename***]

Options

-p ***slot number***	- send only the logical address mappings intended for ***slot*** If ***slot*** is "all", send logical address mappings to all cards.
-b	- background, when set, all messages go to `syslogd` only If not set, messages are printed to standard out.
-d	- print debug messages
-n	- no execution: prints **grinch** commands that would be used to configure the media card, but does not actually execute them
-q	- query the media card for its logical address mappings and print the results
-f ***config_filename***	- get configuration from ***config_filename*** instead of the default, ***etc/grlamap.conf***
-i ***timeout_interval***	- indicates how long to wait for a reply from the media card
-o ***output_filename***	- sends all output to ***output_filename***

Default configuration file

`/etc/grlamap.conf`

This is the default configuration file for the HIPPI logical address configuration.
The name of another configuration file can be specified as the [**-f** ***config_filename***] option.
A file name of – *(hyphen)* specifies that the entries will be read from standard input.

grmaint
(CLI+shell)

Sends a hand-coded maintenance packet to a port.

Permission level: system

Syntax

grmaint [-**B**] [**-c** ***count***] [**-d**] [**-h** ***hwtype***] [**-i** ***interval***] [**-p** ***portcard***] [**-q**]
[**-v**] **value** [. . .]

Options

-B	- specifies that the value command-line arguments (following the options) will specify the values of individual bytes in the packet, not entire words. Arguments for each byte must be presented in network byte order.
-c *count*	- specifies that count copies of the packet will be sent to the destination media card. By default, the packets will be sent one second apart.
-d	- specifies the debug option, which currently does nothing
-h *hwtype*	- specifies that the value of the hardware type field in the GRID header should be set to the value for **hwtype**
-i *interval*	- specifies that each copy of the packet will be sent interval seconds apart
-p *portcard*	- specifies that the packet or group of packets will be sent to the destination portcard
-q	- specifies the quiet option, no information printed
-v	- specifies the verbose option, contents of any unexpected packets received

grmem
CLI+shell

This is a debug-only command.

It is used to access media card memory for debug purposes. Ascend recommends you use this command only under direction of Technical Support staff.

Permission level: system

Syntax

```
grmem  [-B]  [-c count]  [-d]  [-h hwtype]  [-i interval]  [-p portcard]  [-v]
           [-q] mem_request  [...]
         [portcard:]address[,count]
         [portcard:]address=value[[,value]...]
```

portcard	- the media card for which this request is intended
address	- the (starting) address being set or examined, specified in standard C notation: a leading '0x' indicates a hexadecimal number; a leading '0' indicates an octal number; anything else is a decimal number.
count	- specifies that count consecutive words of memory should be examined and their contents displayed, **count** is separated from the address by a comma. (When no count is specified, only a single word of memory is examined and displayed.)
value	- specifies that the contents of address should be set to **value**, which is specified in standard C notation: a leading '0x' indicates a hexadecimal number; a leading '0' indicates an octal number; anything else is a decimal number. **value** is separated from the address by an equal sign (=). A comma-separated list of value specifies that each consecutive word in memory, starting at address, should be set to each consecutive value in the list.

Options

-B	- print the memory values as separate 8-bit bytes, not as complete 32-bit words.
-c *count*	- specifies that count groups of packets will be sent to the destination port card. By default, each group of packets will be sent one second apart.
-d	- specifies the debug option, which currently does nothing.
-h *hwtype*	- specifies that the value of the hardware type field in the GRID header should be set to the value for hwtype.

-i *interval*	- specifies that each copy of the packet (or group of packets) will be sent interval seconds apart.
-p *portcard*	- specifies that the packet or group of packets will be sent to the destination portcard.
-q	- specifies the quiet option. **grmem** will not print any information about received packets.
-v	- specifies the verbose option. **grmem** will print the contents of any unexpected packets received (which are otherwise simply ignored).

grmrflash
(CLI+shell)

This command completely initializes a flash memory device prior to installing software on the GRF control board.

A workaround may be required to use this command due to the lack of space on RAM to store .

Warning: Dangerous, use only in an emergency recovery situation under direction of the Technical Support staff.

Permission level: system

Syntax

grmrflash

If there is not enough space in RAM to load new release files, use this procedure:

1 Log in as root. Put all media cards in hold:

```
# grreset -h all
```

2 Delete the media card binaries to make room in memory:

```
# rm -rf /usr/libexec/portcards
```

Now there will be enough room for the new release files and you can proceed with **grmrflash**.

grms (CLI+shell)

Non-privileged users can use **grms** to shutdown, reboot, or halt a GRF system that uses an RMS node. **grms** can be configured with specific-purpose logins as described below. A man page for **grms** is available.

The primary requirement is that the **grms** command be entered directly from the RMS node or the VT-100. It will not function remotely if entered from another terminal networked to the node. If the site operates in this manner, the **shutdown** command and its corresponding options should be used.

grms -h corresponds to the **-h** option of the **shutdown** command.
grms -s corresponds to **shutdown** with no options.

Enter **grms** without an option to view a short user guide.

Permission level: system

See also: **shutdown**

Syntax

grms [**-h**] [**-r**] [**-s**]

Options

-h	- halts operating system indefinitely
-r	- reboots system
-s	- shuts down the system in an orderly manner

Special logins

If you are working from the VT-100 or similar unit connected to the control board, specific-purpose logins enable non-privileged operators to halt, reboot, or shut down the system :

```
login: halt
login: reboot
login: shutdown
```

To configure these logins, run **adduser** and add these account entries in the password file:

```
halt:*:0:0::0:0:halt:/:/bin/grms-halt
reboot:*:0:0::0:0:reboot:/:/bin/grms-reboot
shutdown:*:0:0::0:0:shutdown:/:/bin/grms-shutdown
```

After running **adduser**, use the **passwd** command to assign passwords to these accounts.

grppp (CLI+shell)

The **grppp** manages a PPP interface and can be used to configure the interface as well as to display interface status.

Warning: Configuration changes made with **grppp** interactively are not maintained across media card reboots. Permanent changes must be made in the `/etc/grppp.conf` file.

Permission level: system

Syntax

grppp [**-n**] [**-f** ***command_file***] [**-i** ***interface_name***]

-n	- enables the no execute option Instead of executing the corresponding **grinch** commands, they are copied to `standard out`. Using this option is an excellent way to verify editing changes to a batch file, such as `/etc/grppp.conf`, without changing the system.
-f ***command_file***	- starts batch mode instead of running interactively, **grppp** reads its input from ***command_file***. When end-of-file is reached, **grppp** exits.
-i ***interface_name***	- execute commands only for the interface ***interface_name***. Commands for any other interface are silently ignored.

Options

help	- displays a synopsis of all the commands
interface ***interface_name***	- attaches **grppp** to a PPP interface. PPP interface names follow the `gx0yz` interface name conventions
quit	- terminates the program
show configuration	- displays the current values of all configurable PPP objects

Interactive commands

You can use a **grppp** command to display interface-specific protocol information. These are the **grppp** status commands:

```
grppp show configuration
grppp show negotiation trace status
grppp show maximum configuration request count
grppp show maximum failure count
grppp show maximum terminate count
grppp show restart timer interval
grppp show lcp keepalive interval
grppp show lcp keepalive packet threshold
grppp show lcp mru
grppp show lcp status
grppp show lqr timer interval
grppp show lqr status
grppp show ipcp status
```

Enter commands in lower case, short forms of words can be used. First, you must declare a specific logical interface with an **interface gx0yz** statement, then the **gx0yz** prompt is generated.

```
# grppp
> interface gs090
gs090>
```

Once you have set the target interface in the prompt, you can enter the desired **show** commands against this interface. Use another **interface** statement to change interfaces.

Here are two examples:

```
# grppp
> interface gs090
gs090> show config
  General Configuration:
    Maximum configure request count: 10
    Maximum request failure count: 5
    Maximum terminate request count: 2
    Negotiation tracing is enabled
    Restart timer interval: 3000 milliseconds
  LCP Configuration:
    Magic number is disabled
    Initial MRU: 1500
    Keepalive interval: 0 milleseconds, disabled
    Keepalive packet threshold: 5
  LQR Configuration:
    LQR is disabled
    Timer interval: 0 milleseconds
  IPCP Configuration:
    enable IPCP
  OSINLCP Configuration:
    disable OSINLCP

gs090> show lcp status
  LCP Status:
    Bad addresses: 0
    Bad controls: 0
    Packets too long: 0
    Bad FCSs: 0
    Local MRU: 1500
    Remote MRU: 1500
  LCP Configuration:
    Magic number is disabled
    Initial MRU: 1500
    Keepalive interval: 0 milleseconds, disabled
    Keepalive packet threshold: 5
```

grreset
(CLI+shell)

This command resets one or more specified media cards. The **grreset** command can be used on a media card without disturbing normal GRF system operations.

The **grreset -D *slot*** command is the suggested way to perform a media card dump.

Permission level: system

Syntax

grreset [**-Ddh**] [**-i *interval***] [**all** | ***slot*** [***slot*** [...]]

interval	-	length of time to wait for message from control board confirming reset
slot	-	chassis slot number, from 0 to 3 (or 0 to 15)
all	-	all media cards are selected to reset

Options

-d	-	turns debug mode on, debug messages are sent
-D	-	dumps memory when media card comes back up
-h	-	holds media card in reset until a second reset command is executed
-i *interval*	-	sets time for arrival of control board reset confirmation message

Examples

You must log in as `root`. The **grreset** command requires that you specify the appropriate media card by its chassis slot number.

To reset all the media cards, enter:

```
# grreset all
```

To reset the media cards in slots 0 and 1, enter:

```
# grreset 0 1
```

To reset the card in slot 4 and dump its memory, enter:

```
# grreset -D 4
```

To reset the card in slot 4 and return debug information, enter:

```
# grreset -d 4
```

grrmb
(CLI+shell)

This command enables you to access an early-development version of the maintenance command set that runs on the GRF control board and a RMS node system's router manager board (RMB).

After **grrmb** is entered, the GR ##> prompt returns where ## is the number of a chassis slot. The default is 66, which specifies that the command will act on slot 66, the control board. To change the default slot, use the maintenance command **port *slot number*** as shown in the example below.

To exit **grrmb**, enter **quit** or **exit**.

Permission level: system

Syntax

grrmb

Example

The following scenario starts the GR##> prompt, displays a list of commands, and then exits **grrmb** to return to the CLI prompt:

```
super> grrmb
GR 66> ?
Ascend Commands
        ?
        bignore
        fan
        maint -[hi|fd] [<port>:]<data> [,<data>[...]]
        power
        port <port>
        temp
GR 66> quit
super>
```

Available commands

?	- lists **grrmb** command set
bignore	- displays broadcast ignore status, on or off
fan	- displays chassis fan RPMs

Here is output for a GRF 400:

```
GR 02> fan
Fan 0 : Curr 75 Last 9 Diff 66
Fan 1 : Curr 229 Last 162 Diff 67
Fan 2 : Curr 179 Last 112 Diff 67
```

Here is output for a GRF 1600:

```
GR 15> fan
Fan 0 : Curr 1 Last 0 Diff 1
Fan 1 : Curr 1 Last 0 Diff 1
```

maint - operates media-specific commands, described in media configuration chapters in *GRF Configuration Guide*

power - displays on/off status of GRF 1600 power supplies (not available on GRF 400):

```
GR 15> power
power {1|2} {on|off}
```

port *number* - sets slot number, 0–3, 0–15

temp - returns current temperatures measured at the control board:

```
GR 66> temp
                    LT     HT     TEMP
                   ----------------------
 Measured Junction     55 C   60 C   26 C
Calc. Ext. Ambient     49 C   54 C   20 C
```

The **reset**, **echo**, **help**, and **ver** commands are not available.

grroute
(CLI+shell)

The **grroute** script initializes and manipulates the *static* routes maintained in the `/etc/grroute.conf` configuration file.

Note: These remote routes are accessible via gateways on networks directly connected to media card interfaces.

By default, **grroute** reads a table of routes from `/etc/grroute.conf` and downloads the routes to all available media cards. When a new media card boots, **grroute** is used to configure the new remote route into the kernel and the kernel then distributes them to currently active media cards.

Note: If GateD is configured and in use, set static routes in a GateD `static` statement. There should be no entries in `grroute.conf`.

More information is available in the **grroute** man page.

Permission level: system

Syntax

grroute [**-n**] [**-f** ***file***] [**-p** ***card*** [...]]

grroute -g ***gatenet*** **-m** ***mask*** [**-n**] [**-f** ***file***] [***dest*** [...]]

grroute -c [**-n**] ***destination netmask gateway***

Arguments

The format of each entry in `/etc/grroute.conf` is:

```
destination   netmask   gateway
```

destination	- the Internet address of the destination host or network (a default route can be specified with a ***destination*** value of `default` or `0.0.0.0`)
netmask	- netmask for the destination (must be `255.255.255.255` for destination hosts)
gateway	- the Internet address of the gateway to which packets destined for ***destination*** will be forwarded (this gateway must be on a network directly attached to a media card interface)

Options

-c	- causes **grroute** to configure a single route from command line arguments instead of reading arguments from `/etc/grroute.conf`

-f ***file***	- causes **grroute** to read the table of routes from ***file*** instead of the default `/etc/grroute.conf` A file name of "-" specifies the table will be read from standard input.
-g ***gatenet***	- specifies that **grroute** configure only those routes whose gateway is on a directly-attached network ***gatenet*** Use with **-m** ***mask*** to specify the network mask for ***gatenet***.
-m ***mask***	- used with **-g** ***gatenet*** to specify the network mask for ***gatenet***
-n	- prints the **route** commands that would be used to configure the kernel or media cards but does not execute them

grrt
(CLI+shell)

The **grrt** command can be used to examine individual entries on a specific media card, or to examine the entire route table on a media card.

Note: During normal operations**, grrt** is not recommended for route table configuration because it does not ensure that route tables are synchronized among the various media cards. Without this coordination, the router is not likely to forward traffic correctly.

The recommended method for configuring routes is to use the `/etc/grifconfig.conf` and `/etc/grroute.conf` files.

See also: **traceroute**, **netstat**

Permission level: system

Syntax

grrt **-a** [**-c** ***count***] [**-d**] [**-h** ***hwtype***] [**-i** ***interval***] [**-p** ***slot***] [**-v**]
address netmask nexthop dest_if [***metric***]

grrt **-A** [**-c** ***count***] [**-d**] [**-h** ***hwtype***] [**-i** ***interval***] [**-p** ***slot***] [-v] ***nexthop dest_if***

grrt -[**rs**] [**-c** ***count***] [**-d**] [**-h** ***hwtype***] [**-i** ***interval***] [**-p** ***slot***] [**-v**] ***address***

grrt -[**RSZ**] [**-c** ***count***] [**-d**] [**-h** ***hwtype***] [**-i** ***interval***] [**-p** ***slot***] [**-v**]

grrt [-S] [**-p** ***slot***]

Arguments

The **grrt** command interprets command line-arguments as follows:

address	- IP address of the destination host or network A destination host is indicated by specifying a netmask of all ones.
netmask	- netmask of the destination address A destination host address is indicated by specifying a netmask of 255.255.255.255 (all ones).
nexthop	- IP address of the system on a directly-attached network to which the packet should be forwarded
dest_if	- GRIT (GR Internal Transmission) address of the media card/interface to which packets intended for the destination address should be routed (see **grit** man page for format of GRIT addresses)

metric	- a flag used to indicate special cases: **0** - normal host or network route **16** - internal route to the operating system (the value of *nexthop* is ignored) **32** - route to a network directly attached to the router (value of *nexthop*, if any, is ignored)

grrt command options

The **grrt** command supports the following options that specify the configuration command to be performed and the command-line arguments expected:

-s *address*	- shows (displays) the route table entry in a media card routing table that would be used for a packet destined for *address*
-S	- shows (displays) a media card routing table
-Z	- deletes all routes from a media card routing table

Functional options

The **grrt** command supports the following options:

-c *count*	- specifies that *count* copies of the command will be sent to the destination media card (by default, each copy will be sent one second apart)
-d	- specifies the debug option (not available)
-h *hwtype*	- specifies that the value of the hardware type field in the GRID header should be set to the value for *hwtype*
-i *interval*	- specifies that each copy of the command will be sent *interval* seconds apart
-p *slot*	- specifies that the command will be sent to the destination media card
-q	- specifies the quiet option in which **grrt** does not print any information about received packets
-v	- specifies the verbose option in which **grrt** prints the contents of any unexpected packets received (such packets are otherwise ignored)

Deprecated options

-a *address netmask nexthop dest_if* [***metric***]

- adds a route to a media card route table but does not change the kernel, changes are lost at media card reset (similar to UNIX **route** command)

-A *nexthop dest_if*	- adds a default route to a media card routing table
-r *address*	- deletes a route from a media card routing table
-R	- deletes the default route from a media card routing table

Example

To view the routing table for the media card in slot 1, enter:

```
#  grrt -p 1 -S
```

Here is an example of the type of information returned:

```
# grrt -S -p 1
  default                         0    0.0.0.0          inx 0      UNREACH
  0.0.0.0          255.255.255.255  1    0.0.0.0          inx 0      DROP
  10.20.1.0        255.255.255.0    22   0.0.0.0          ge034      FWD
  10.20.1.133      255.255.255.255  21   0.0.0.0          ge034      LOCAL
  10.20.1.255      255.255.255.255  20   0.0.0.0          ge034      BCAST
  10.20.2.0        255.255.255.0    10   10.205.1.150     ga00f0     FWD
  10.205.1.0       255.255.255.0    9    0.0.0.0          ga00f0     FWD
  10.205.1.133     255.255.255.255  8    0.0.0.0          ga00f0     LOCAL
  10.205.1.255     255.255.255.255  7    0.0.0.0          ga00f0     BCAST
  192.0.0.0        255.0.0.0        5    0.0.0.0          inx 0      DROP
  192.0.0.1        255.255.255.255  5    0.0.0.0          inx 0      DROP
  192.168.11.0     255.255.255.0    6    206.146.160.1    inx 0      RMS
  192.168.8.0      255.255.255.0    16   0.0.0.0          gf020      FWD
  192.168.8.133    255.255.255.255  15   0.0.0.0          gf020      LOCAL
  192.168.8.255    255.255.255.255  14   0.0.0.0          gf020      BCAST
  192.168.10.0     255.255.255.0    24   10.205.3.150     ga00f1     FWD
  192.168.10.0     255.255.255.0    24   10.205.3.150     ga00f1     FWD
  192.168.1.1      255.255.255.255  25   204.101.9.150    ge031      FWD
  192.168.1.2      255.255.255.255  23   205.1.1.2        inx 0      ATMP
  192.168.160.0    255.255.255.0    3    0.0.0.0          inx 0      RMS
  192.0.0.0        240.0.0.0        1    0.0.0.0          inx 0      DROP
  192.0.0.0        255.0.0.0        3    0.0.0.0          inx 0      RMS
  192.0.0.0        255.255.255.255  3    0.0.0.0          inx 0      RMS
  255.255.255.255  255.255.255.255  2    0.0.0.0          inx 0      BCAST
#
```

grsavecore
(CLI+shell)

The **grsavecore** command copies and formats information generated when a kernel panic occurs. The data is written to standard output. Normally it is used to copy data out of flash or a swap partition (similar to **savecore**), or to pretty-print that data.

Permission level: system

Syntax

grsavecore [**-h**] [**-d**] [**-f**] [**-n**] [**-r**] [**-C** [**-t**] [**-i** ***inputfile***]

Options

-h	- causes a brief usage message to be printed, the program then exits without performing any other action
-d	- sets debug option enables the printing of a little extra diagnostic information to `stderr` as the program runs
-f	- sets format option to cause the dump data to be formatted so that it human-readable for humans (default)
-i ***inputfile***	- specifies the filename to read as raw input data. By default, **grsavecore** reads data from from the dump device (normally a partition, `/dev/wd0b`, on a flash disk). This is useful for formatting a file that was previously generated by running this program with the **-r** flag.
-n	- generate no output, sometimes useful with -C option
-r	- sets raw option, causes dumped data to be output without any alteration or formatting. This is useful when for copying data from the dump device to a normal file system.
-C	- sets the clear flag to cause the header block on the data source (either default or specified with -i) to be overwritten. This is most useful for clearing out old dumps from the dump device. The block is only clobbered after the data has been successfully read and output.
-t	- sets stack trace residue for use with **gdb**

grsite
(CLI+shell)

grsite saves specified files that are not located in the `/etc` directory. **grwrite** saves all the files in the `/etc` directory.

The **grsite** command manages special site-specific images, creating a site file that contains the special images. These images are extra files that are installed after the main release is moved to the RAM drive. The site file overlays the system and configuration files, and in this way allows special files to be easily incorporated into a configuration and just as easily removed.

grsite preserves the site file over a system boot. The **grsite --perm** option preserves the site file across a **grfins** installation or upgrade.

grsite can be useful for installing a single binary image, such as a `/var/portcards/fddi.run` binary, that is to be used for debugging or testing. The **grsite** command provides options to add, delete, and list files in either the current release set, the next boot release set, or an arbitrary release set. Note that the command works relative to the root of the file system unless told explicitly to do otherwise.

To add the `fddi.run` binary to the currently-running system:

```
# cd /var/portcards
# grsite fddi.run
```

Note that **grsite** provides the `/var/portcard` prefix when it copies the ***pattern*** (`fddi.run`) into the site file.

See also: **grwrite**, **grsnapshot**

Permission level: system

Syntax

grsite ***options pattern***

pattern	- a file name prefix or file name

Options

--delete	- indicates that the ***pattern***(***s***) should be removed from the given release
--list	- lists the given ***pattern*** or, if no ***patterns*** are specified, displays all files in the site file archive Note that duplicate entries may exist if a file has been added (that is, overwritten). The last entry is the one that will get used.
--next	- uses the next-boot version instead of the current version
--nopath	- indicates that the command should not fill in the current directory name in front of the file patterns

		The expectation is that the user would then fill in the complete relative path on each pattern. For example, to install `/home/me/var/portcards/fddi.run`, the user would **cd** to `/home/me` and issue a **grsite** command with the **--nopath** option and a pattern of `var/portcards/fddi.run`.
--perm	-	preserves the site file across an upgrade or installation (**grfins**)
--target	-	specifies the target device, the current device names are **internal** and **external**
--version	-	specifies an explicit version instead of using the next boot or the current boot.
--help	-	provides a short help message

Examples

grsite can create two files (databases):

– one to save changes during a reboot

– one to save changes when upgrading

For example:

`1.4.x,default.site.gz`- database used to retain changes when rebooting
`1.4.x,default.sitep.gz`- database used to retain changes when upgrading

To view contents in the `1.4.x,default.site.gz` file:

```
# grsite --list
```

To view contents in the `1.4.x,default.sitep.gz` file:

```
# grsite --list --perm
```

If you wish to **grsite** a directory and its contents, change to the parent directory and execute **grsite** on directory name. For example, enter:

```
# grsite foo
```

where `foo` is a subdirectory off `root` (/`foo`)

Note: If you delete a file or directory from the RAM file system before executing the **grsite --delete** command, you can **tar gunzip** it from `/flash`. This is to prevent rebooting the router to restore defaults.

For example, if you deleted the directory `/foo` with contents before executing **grsite**, execute the following command to view contents of the tar file:

```
gzcat /flash/1.4.x,default.site.TAR.gz | tar tf -
```

results from gzcat:

```
/foo
/foo/file1
/foo/file2
```

```
/foo/dir
/foo/dir/file3
```

where `1.4.x,default.site.TAR.gz` is the site file to restore defaults.

Look for files or directory to extract, go to parent directory on the RAM file system and execute:

```
gzcat /flash/1.4.x,default.site.TAR.gz | tar tf - /foo
```

Caution: Permanent site files are not carried over when switching from one version to another using **setver** command. Permanent site files are carried over when using **grfins**.

grsnapshot (CLI+shell)

This command is for GRF systems with new control boards. The equivalent command for RMS node-based systems is **grc**.

The **grsnapshot** command saves configurations and release images to the internal or external flash device, or to another file system. It is used to copy or create another version of the configuration files or to make a complete backup of the flash device. This script will back up, copy, and save configuration files to either a new version name or the same version name on a different flash device.

To make an exact, bootable copy of the internal flash device to the external flash device, **grsnapshot** is used with the **dup** option. This command initialize the external flash device, writes a file system to it, writes the boot blocks to the device, and copies all the files and directories from the internal flash drive to the external flash drive.

See also: **grwrite**, **grsite, grc**

Permission level: system

Syntax

grsnapshot **-s**[**i** | **e** | **d** | **u**]=***revision,version*** **-d**[**i** | **e** | **d** | **u**]=***revision,version***
[**--dup=ie**] [**--dup=ei**] [**--force**]

revision,version	- specifies the system software release and version name, as in `1.4.x,default`

Options

-d	- destination of the software to be archived
-s	- source of the software to be archived
e	- specifies the external flash device to copy from/to: `/dev/wd1a`
i	- specifies the internal flash device to copy from/to: `/dev/wd0a` (default)
d	- specifies the directory to copy from/to
u	- specifies the URL to copy from/to
--dup=ie	- duplicates the internal (**i**) device on the external (**e**) device
--dup=ei	- duplicates the external (**e**) device on the internal (**i**) device
--force	- used with **--dup**, requires a copy or overwrite to be done if a file system exists on the destination device

Examples

The following command copies the `1.4.x,default` system on the internal flash to the external flash device as `1.4.x,default`:

```
# grsnapshot -si=1.4.x,default -de=1.4.x,default
```

The following command makes a copy of the `1.4.x,default` system on the internal flash to the file name `1.4.x,bob` on the internal flash:

```
# grsnapshot -si=1.4.x,default -di=1.4.x,bob
```

The following command copies the `1.4.x,default` system on the internal flash to a gzipped tar file with the name `/tmp/mybackup.tar.gz` (this option not currently available):

```
# grsnapshot -si=1.4.x,default -dd=/tmp/mybackup.tar.gz
```

The following command copies or overwrites the configuration contained in the file `/tmp/mybackup.tar.gz` onto the internal flash device:

```
# grsnapshot -sd=/tmp/mybackup.tar.gz -di
```

The following command creates an alternate configuration on the internal flash based upon the currently-running configuration on the internal flash device:

```
# grsnapshot -si -di=revision,version
```

The following command backs up the internal flash device to the external device. It initializes the external device, writes a file system to it, writes the boot blocks to the device, and copies all the files and directories from the internal flash device to the external flash device:

```
# grsnapshot --dup=ie
```

The following sequence illustrates how the **--force** option is applied:

```
# grsnapshot --dup=ie
```

If a file system exists on the destination device, you will receive a message to that effect.
To enable the overwrite, use the **--force** option:

```
# grsnapshot --dup=ie --force
```

grstat
(shell)

Layer 3 statistics

The **grstat** command reports layer 3 (IP and ICMP) forwarding statistics for all media card types. Error reporting includes the saved source and destination IP addresses of the packet that caused the last error of each type reported.

Layer 2 statistics

The **grstat** command reports many of the Layer 2 (data link layer) statistics currently reported by individual media card **maint** commands. Examples are at the end of this section.

Syntax

grstat [-ahltvzZ] [-w *width*] [-c *count*] [-i *seconds*] *stat_type* [*card* | *interface* . . .]

stat_type specifies the type of statistics to retrieve. Types include:

grid	- displays the combus and gridax statistics.
ipstat	- lists IP statistics.
ipdrop	- lists IPDROP statistics.
ip	- lists IPSTAT and IPDROP statistics.
icmpin	- lists ICMPIN (input) statistics.
icmpout	- lists ICMPOUT (output) statistics.
icmperr	- lists ICMPERR (error) statistics.
icmp	- lists ICMPERR, ICMPIN, and ICMPOUT statistics.
l2	- displays the media I/O statistics, **grstat l2rx** displays receive side statistics, **grstat l2tx** displays transmit side statistics.
switch	- displays the switch statistics.
all	- lists all of the above statistics.

Options

card	- specifies the slot number of the media card to query.
interface	- specifies the logical interface name in the format `gx0yz`.

The *interface* and *card* options can be combined. However, if a logical interface is specified twice as a result of being on a specified card and by being specified as an interface, that interface is counted twice when values are totalled.

-a	- enables statistics lines with a count of zero to be printed out, overriding the default case in which they are not. A second **-a** also prints zeroed IP addresses instead of blanks in output displays that have IP addresses.
-h	- help function, prints the option summary, including the complete list of supported ***stat_type*** values.
-l	- lists out data per individual interface. By default, if a card is specified, the data shown is a summary total for all interfaces found on the card.
-t	- displays totals for the statistics requested for specified interfaces after the interfaces are listed. This option is not necessary to obtain totals for an entire card. It can be used in conjunction with **-l** to view a total at the bottom, below the list of individual interfaces .
-v	- displays additional debugging or error information, generally useful only for debugging.
-z	- zeroes **stat_type** statistics on the specified card or interface.
-Z	- prints out statistics for the specified interface(s) and then zeroes them. This option can be used in conjunction with **-c**.
-w *width*	- specifies display in ***width*** number of characters. A line width can be set greater than 80 characters to improve viewing truncated statistics descriptions.
-c *count*	- displays the specified statistics ***count*** number of times at five-second intervals, the default (default interval is modified with **-i**).
-i *seconds*	- specifies an interval in seconds, the default is five seconds, used in conjunction with **-c.**

Beware that statistics gathering consumes communications bus bandwidth, avoid using a shorter interval than the default.

Return values

grstat returns "`-1`" if invalid options or other serious errors occur, 0 otherwise. In general. **grstat** tries to keep going when it has trouble talking to what are otherwise supposed to be valid interfaces.

Layer 3 examples

List all IP statistics on the GRF:

```
# grstat ip
all cards (19 interfaces found)
  ipstat totals
       count description
  1022766098 packets received
     1035601 packets dropped
  1021730497 packets forwarded
  ipdrop totals
       count description
     1035601 no route to destination address
```

List ICMPERR statistics for the system:

```
# grstat icmperr
all cards (19 interfaces found)
  icmperr totals
       count error
     1265253 too much ICMP; type throttled
        7284 no route back to originator
```

List ICMPERR statistics totals on card 5 and individually for interfaces `gf0b0` and `gf0b1`:

```
# grstat icmperr 5 gf0b0 gf0b1
```

List IPSTAT statistics totals for all interfaces on the GRF:

```
# grstat ipstat
```

List IPSTAT statistics individually for all interfaces on the GRF:

```
# grstat -l ipstat
```

List IPSTAT statistics individually for all interfaces on the GRF and then list the totals for all interfaces found:

```
# grstat -l -t ipstat
(0 interfaces found)
  ipstat totals
       count description
gs020
  ipstat
       count description
gs0280
  ipstat
       count description
ge030
  ipstat
       count description
ge031
  ipstat
       count description
ge032
  ipstat
       count description
ge033
  ipstat
       count description
```

```
    ge034
      ipstat
           count description
    ge035
      ipstat
           count description
          198678 packets received
            5601 packets dropped
    ge036
      ipstat
           count description
    ge037
      ipstat
           count description
    all cards (5 interfaces found)
      ipstat totals
           count description
          198678 packets received
            5601 packets dropped
```

List IPSTAT statistics on `gf0b3`, clear them, wait 60 seconds, and repeat a total of 100 times:

```
    # grstat -Z -c 100 -i 60 ipstat gf0b3
```

This is a sample of IP counts and statistics covered by **grstat**:

```
# grstat -a -w 70 ip ga00f1
ga00f1
  ipstat
       count description
        8416 total packets received
           0 packets dropped
           0 packets forwarded normally
           0 packets redirected out receiving interface
           0 packets fragmented
           0 fragments created
           0 packets forwarded locally to card
           0 packets handled by the card
        8416 packets forwarded to the RMS
           0 packets segmented to the RMS
           0 multicast packets received
           0 multicast packets attempted to route
           0 multicast packets sent to RMS
           0 multicast packets received on rincorrect interface
           0 multicast packets for forwarding
           0 multicast packets (copies) transmitted
           0 packets ATMP encapsulated
           0 packets ATMP decapsulated
ipdrop
        last            last
count   source addr    dest addr  reason
0                                 packet received on down interface
```

```
0                                  received data below IP header minim
0                                  not version 4 IP
0                                  header length below minimum
0                                  bad header checksum
0                                  header length longer than packet le
0                                  received data less than packet leng
0                                  source address from 127.*.*.*
0                                  source address from 192.0.2.*
0                                  packet filtered on input from media
0                                  no route to destination address
0                                  destination interface is down
0                                  TTL expired
0                                  needed to frag packet but DF set
0                                  multicast routing not yet supported
0                                  routing to bridge groups not suppor
0                                  route table says to drop dest addre
0                                  unknown special processing code
0                                  IP option with length <= 0
0                                  IP option length past header length
0                                  Record Route offset less than minim
0                                  couln't find interface address for
0                                  TimeStamp offset less than minimum
0                                  bad TimeStamp option flag
0                                  TimeStamp option overflow beyond ma
0                                  Source Route offset less than minim
0                                  next Strict Source Route address no
0                                  bad ICMP checksum
0                                  local interface is down for ICMP EC
0                                  no route back for reply to ICMP ECH
0                                  no next route for Strict Source Rou
0                                  no next route for Loose Source Rout
0                                  no interface addr for Source Route
0                                  can't forward link-layer broadcast
0                                  can't forward link-layer multicast
0                                  loose source routing disabled
0                                  strict source routing disabled
0                                  no buffer to generate packet
0                                  Rate overflow on ICMP generation
0                                  packet filtered into-me
0                                  ATMP err: can't find Home Network e
0                                  ATMP err: bad GRE header
0                                  ATMP err: can't find mobile node en
0                                  ATMP err: Invalid IP header
0                                  multicast replicated, original drop
0                                  multicast TTL expired
0                                  multicast packet on wrong interface
```

Layer 2 options and examples

```
# grcard
 0       SONET_V1   running
 2        FDDI_V2   running
# grstat grid
card 0
  GRID statistics
             count description
            141153 COMBUS messages received
            140892 COMBUS GRID messages received
                 2 COMBUS receive long messages
               261 COMBUS GRIT messages for TX-CPU
           5255261 COMBUS last status register
                 1 COMBUS time in
               194 GRIDAX packets received
                 4 GRIDAX restart
                24 GRIDAX acks received
                 5 GRIDAX requested acks received
                29 GRIDAX control packets received
               165 GRIDAX output queued
               187 GRIDAX packets sent
                22 GRIDAX acks sent
                22 GRIDAX control packets sent
card 1
error -1 reporting data
card 2
  GRID statistics
             count description
             28041 COMBUS messages received
             28041 COMBUS GRID messages received
             27797 COMBUS GRID echo requests
                 4 COMBUS receive long messages
               207 GRIDAX packets received
                 5 GRIDAX restart
                27 GRIDAX acks received
                 2 GRIDAX requested acks received
                29 GRIDAX control packets received
               178 GRIDAX output queued
               204 GRIDAX packets sent
                26 GRIDAX acks sent
                26 GRIDAX control packets sent
card 3
error -1 reporting data
```

Layer 2 statistics are reported for the entire card. If slots are empty or cards are not responding, error messages are generated.

```
# grstat l2
card 0
  Layer 2 statistics
    physical port 0
```

```
               count description
               33418 TX packets
              467852 TX bytes
card 1
  Layer 2 statistics
Got error -1 - no response from 0:0x1:0 (card 1)
error -1 reporting data
card 2
  Layer 2 statistics
Got error -6 - no response from 0:0x2:0 (card 2)
error -6 reporting data
card 3
  Layer 2 statistics
Got error -1 - no response from 0:0x3:0 (card 3)
error -1 reporting data
```

Layer 2 statistics are not currently gathered from each type of media card.

```
# grstat switch
card 0
  Switch statistics
               count description
               33434 RX packets
             1872304 RX bytes
                   1 Switch receiver reset
card 1
error -1 reporting data
card 2
error -6 reporting data
card 3
error -1 reporting data
```

grwrite
(CLI+shell)

The **grwrite** command saves changes made to the `/etc` directory on the RAM file system over to the internal flash device so the changes are available during the next boot.

Specifically, the **grwrite** command saves the `/etc` directory to flash so the `/etc` configuration files and configuration changes are made permanent, and also preserves symbolic links created within the `/etc` directory. It is critical to save configuration file changes intended to be permanent. Changes made to files other than those in `/etc` must be saved using **grsite**.

When installing a new software version, make sure all changes to files in `/etc` that you intend to be carried forward to the new release have been saved to flash. Execute **grwrite** -before- you begin a software upgrade.

If you execute the **grwrite** command after doing a **grfins**, the changes are only saved to the currently-running version. The currently-running version is assigned by **grwrite** to be `default`.

grwrite uses the **-r** ***revision*** option as the revision structure to copy the files to. If **-r** ***revision*** is not specified, the revision information is retrieved by the **getver** command.

Note: Any changes you make to the `/etc/services` file are overwritten when you install a new software release. Record these changes and add them back after the upgrade.

See also: **grsnapshot**, **grsite**

Permission level: system

Syntax

grwrite [**-e]** [**-i**] [**-A**] [**-B**] [**-d** [***device***]] [***filename***] [**-n**] [**-r** ***revision***] [**-v**]

Options

The **grwrite** command supports the following options:

-e	- uses the device descriptor for the external flash device: `/dev/wd1a`
-i	- uses the device descriptor for the internal flash device: `/dev/wd0a` (this is the default)
-A	- (option not available) uses the device descriptor for the external flash device in PCMCIA slot A, `/dev/wd1a`
-B	- (option not available) uses the device descriptor for the external flash device in PCMCIA slot B, `/dev/wd2a`
-d ***device_descriptor***	- uses the specified device descriptor, this option enables another device to be acted upon

filename	- **grwrite** saves the specified ***filename*** to flash. Specify a file in the /etc directory
-n	- Specifies the no execute mode. In this mode, **grwrite** checks to see if there are files that need updating but does not save them. If you use the **-vn** option, **grwrite** lists files currently unsaved in the `/etc` directory.
-r *revision*	- specifies the software release and version against which to perform the **grwrite**. This option is in the form *release,version*.. For example, `1.4.x`,`default` or `1.4.x`,`atmtestB`.
-v	- Specifies verbose mode so that **grwrite** lists the files that are being updated in, added to, and/or removed from flash memeory. If you use the **-vn** option, **grwrite** lists files currently unsaved in the `/etc` directory.

gsm
(CLI + shell)

The GateD State Monitor (GSM) is an interactive interface used to query a running GateD daemon about internal **gated** variables. Commands include protocol-specific (IP, OSPF, BGP, RIP, IS-IS) queries for memory, route table, interface list, and other internal parameters. You can start and use the **gsm** command in the CLI or at the UNIX prompt, both methods are described below. The difference between the two is how you access and start the GSM interface. After you start GSM, the commands are the same for both.

Refer to Chapter 2 in this manual for more information about GSM.

Permission level: system

CLI access and syntax

gsm [***hostname***]

When you enter GSM from the CLI prompt, **gsm** returns information about the GateD running on that local machine. To obtain GateD statistics for a different GRF system, you must access that GateD through the UNIX shell, shell usage is described below.

To access the GSM interface, you need the password from the administrative 'netstar' account, the default password shipped with your system is "NetStar". If you have changed the administrative password, use that new one.

```
super> gsm
Trying 198.xx.xx.1...
Connected to 198.xx.xx.1.
Escape character is '^]'.
Password?-------- ( for example, NetStar )
```

You will see the GSM interface starting, notice that the **gsm** prompt uses the machine domain name:

```
1GRF Gated State Monitor. Version GateD R3_5Beta_3; CVS Branch:A1_4_1;
Path:/
/gated/code/GSM
GateD-router.sitename.com>
```

As shown in the following lines, the CLI help function does not provide descriptions of **gsm** subcommands, you must access **gsm** to view that information:

```
super> ? gsm
Usage: gsm [hostname]
super> gsm ?
usage: telnet [-l user] [-a] host-name [port]
super>
```

Top-level commands
When you have the **gsm** prompt, use **help** to list the top-level commands:

```
GateD-router.sitename.com> help
1GRF HELP: The possible commands are:
1GRF    ?     : Print help messages
1GRF    help  : Print help messages
1GRF    show  : Show internal values
1GRF    quit  : Close the session
1GRF    enable: Enable the session
1GRF    exec  : Execute actions (must be enabled)
1GRF    exit  : Close the session
```

Refer to the Shell example section below for the next levels of **gsm** commands.

Shell usage and syntax

gsm [help] [option]

From a UNIX shell you open a telnet connection on TCP port 616 to the machine running GateD. You can telnet from the administrative LAN or from the GateD machine itself.

To telnet to the GSM interface, you must use the password from the administrative 'netstar' account, the default password shipped with your system is "NetStar". If you have changed the administrative password, use that new one. After user password identification, GSM answers queries from its subcommands.

If GateD is running on 192.22.22.22, this command connects to port 616:

```
telnet -a  192.22.22.22 616
Trying 192.22.22.22...
Connected to 192.22.22.22.
Escape character is '^]'.
Password? -------   ( for example, NetStar )
1GRF Gated State Monitor. Version GateD R3_5Beta_3;
Path:
GateD-router.sitename.com>
```

Note that the **gsm** prompt contains the machine domain name.

Shell example

Top-level commands
With the connection established, use **help** to look at the top-level command options:

```
GateD-router.sitename.com> help
1GRF HELP: The possible commands are:
1GRF    ?     : Print help messages
1GRF    help  : Print help messages
1GRF    show  : Show internal values
1GRF    quit  : Close the session
1GRF    enable: Enable the session
1GRF    exec  : Execute actions (must be enabled)
1GRF    exit  : Close the session
```

Second-level commands

Use the **show** command to look at the display options:

```
GateD-router.sitename.com> show
1GRF HELP: The possible show subcommands are:
1GRF    version  : Show the current GateD version
1GRF    kernel   : Show the Kernel support
1GRF    iso      : Show the ISO support
1GRF    interface [name|index]: Show interface status
1GRF    memory   : Show the memory allocation
1GRF    ip       : Show info about IP protocol
1GRF    task     : Show list of active tasks
1GRF    ospf     : Show info about OSPF protocol
1GRF    timer    : Show list of timers
1GRF    bgp      : Show info about BGP protocol
1GRF    rip      : Show info about rip protocol
1GRF    isis     : Show info about ISIS protocol
```

Third-level

Now look at information options for a specific protocol, in this case, IP:

```
GateD-router.sitename.com> show ip
1GRF HELP: The possible subcommands are:
1GRF    sum       Show info about IP routes
1GRF    exact     [x.x.x/len]: Show info about specific IP routes
1GRF    aggregate [x.x.x/len]: Show info about specific Aggregate routes
1GRF    all       Show entire FIB
1GRF    consume   [x.x.x/len]: Show consuming route and more specic route
1GRF    lessspec   Show less specific routes per protocol
1GRF    refines    Show more specific routes per protocol
```

An abbreviation is accepted provided it unambiguously identifies an entity. Look at the IP summary:

```
GateD-router.sitename.com> sh ip sum
1GRF IP radix tree: 38 nodes, 20 routes
```

The recommended way to view IP route tables is through GSM. Establish a GSM session and use the `show ip all` command:

Display the IP route table:

```
GateD-router.sitename.com> sh ip all
1GRF Sta        78.78.78/24 203.3.1.153     IGP (Id 1)
1GRF Sta        99.99.98/24 202.1.1.153     IGP (Id 1)
1GRF Sta        99.99.99/24 212.1.3.152     IGP (Id 1)
1GRF Sta             127/8  127.0.0.1       IGP (Id 1)
1GRF Sta      198.174.11/24 206.146.160.1   IGP (Id 1)
1GRF Dir         202.1.1/24 202.1.1.151     IGP (Id 1)
1GRF Dir         202.1.2/24 202.1.2.151     IGP (Id 1)
1GRF Dir         202.1.3/24 202.1.3.151     IGP (Id 1)
1GRF Dir         202.5.2/24 202.5.2.151     IGP (Id 1)
1GRF Dir         202.5.4/24 202.5.4.151     IGP (Id 1)
```

```
1GRF Dir         203.3.1/24 203.3.1.151     IGP (Id 1)
1GRF Dir        204.10.1/24 204.10.1.151    IGP (Id 1)
1GRF Sta   204.100.1.147/32 208.1.1.152     IGP (Id 1)
1GRF Sta     205.2.4.138/32 212.1.2.134     IGP (Id 1)
1GRF Dir     206.146.160/24 206.146.160.151 IGP (Id 1)
1GRF Dir         208.1.1/24 208.1.1.151     IGP (Id 1)
1GRF Dir         212.1.1/24 212.1.1.151     IGP (Id 1)
1GRF Dir         212.1.2/24 212.1.2.151     IGP (Id 1)
1GRF Dir         212.1.3/24 212.1.3.151     IGP (Id 1)
```

Now look at a specific set of routes:

```
GateD-router.sitename.com> sh ip cons 202/8
1GRF Dir         202.1.1/24 202.1.1.151     IGP (Id 1)
1GRF Dir         202.1.2/24 202.1.2.151     IGP (Id 1)
1GRF Dir         202.1.3/24 202.1.3.151     IGP (Id 1)
1GRF Dir         202.5.2/24 202.5.2.151     IGP (Id 1)
1GRF Dir         202.5.4/24 202.5.4.151     IGP (Id 1)
GateD-router.sitename.com> quit
```

telnet syntax

telnet [**-l** ***user***] [**-a**] ***host-name*** [***port***]

telnet options

host-name	- name or IP address of machine running **gated**
port	- TCP port 616 to access GSM
-l *user*	- if the remote system understands the **environ** option, user is sent to the remote system as the value for the variable *`user`*
-a	- specifies the automatic telnet login option

help or ?
(CLI)

help and its alias **?** return a list of all registered commands authorized by user's security profile. Use **help** or **?** followed by a specific command name to obtain description or usage information.

Permission level: user

Syntax

help, **?**

help ***command-name***, **?** ***command-name***

Example

```
super> help ls
ls profile-type [ profile-index ] [ field-name [ field-index ]
    [ sub-profile-name ] ... ]
Usage:  view a field or group of fields from the specified profile

ls . [ field-name [ field-index ] [ sub-profile-name ] ... ]
Usage:  view a field or group of fields from the working profile,
same as get command
```

ifconfig
(CLI+shell)

This command is modified for GRF use to assign addresses, mask, and other network parameters to a logical interface. Also, when no optional parameters are specified, **ifconfig** displays the current configuration for a network interface. If a protocol family is specified, **ifconfig** reports only the details specific to that protocol family. To use **ifconfig** to modify the configuration of a network interface, you must be logged in as super user.

If you log in to a GRF as `root`, you automatically get the CLI shell. In the CLI, `root` is super user, hence the `super>` prompt.

Permission level: system

Syntax

ifconfig ***interface address_family*** `[` ***address*** `[` ***dest_addr*** `]]` `[`***parameters***`]`
ifconfig ***interface*** `[`***protocol_family*** `]`

interface	- GRF interface name in format `gx0yz` ATM interface names look like: `ga037f, ga0281, ga01ff` FDDI names: `gf030, gf021, gf012, gf003` HIPPI names: `gh030` (only the slot # changes) HSSI names: `gs030, gs021, gs0180` SONET OC-3c: `go030` (only the slot # changes) 10/100Base-T interface names:`ge030, ge026, ge017` T1 interface names:`gm070–gm079` (the T1 card in slot 7)
address_family	- specifies the address family which affects interpretation of the remaining parameters. Since an interface can receive transmissions in differing protocols with different naming schemes, specifying the address family is recommended. The address families currently supported are inet, iso, ns, and grit. A grit address is specified `cage:card:interface`
address	- IP address, ISO address (for PPP)
dest-addr	- specifies the address of the correspondent on the other end of a point-to-point link
protocol_family	- specifiesprotocol families currently supported are inet, iso, ns, and grit
parameters	- the following parameters can be set with **ifconfig**: alias, -alias, arp, broadcast, debug, -debug, delete, dest_address, down, ipdst, linktype, metric, mtu, netmask, nsellength, ptp, -ptp, trailers, -trailers, link[0-2], -link[0-2], up, proxy, -proxy
alias	- specifies an additional or alias network address for the specified interface
-alias	- deletes the specified alias

arp	- enables ARP between network-level addresses and link level addresses (default), currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.
-arp	- disables ARP
broadcast	- (inet only) specifies the address to use to represent broadcasts to the network, the default broadcast address is the address with a host part of all 1s
debug	- enables driver-dependent debugging code, usually turns on extra console error logging
-debug	- disables driver-dependent debugging code
delete	- removes specified address
dest-addr	- specifies the address of the correspondent on the other end of a point-to-point link
down	- specifies an interface as down, the system will not attempt to transmit messages through that interface.
ipdst	- specify an Internet host who is willing to receive IP packets encapsulating NS packets bound for a remote network An apparent point-to-point link is constructed, and the address specified will be taken as the NS address and network of the destination.
linktype ***type***	- specifies the type of link to be **type**. More common types are PPP, CHDLC, and Frame Relay. Only point-to-point interfaces support setting the link type. Some types understood by **ifconfig** may not be compiled into or understood by the kernel.
link[0-2]	- enables special processing of the link level of the interface. The three link options (0-2) are interface specific in actual effect; however, they are in general used to select special modes of operation.
-link[0-2]	- disable special processing at the link level with the specified interface.
metric ***n***	- sets the routing metric of the interface to **n**, default is 0. The routing metric is used by the routing protocol **routed**. Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host
mtu ***n***	- sets the maximum transmission unit of the GRF interface

netmask ***mask***	- specifies how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the sub-net part, which is taken from the host field of the address.
nsellength ***n***	- (iso only) specifies a trailing number of bytes for a received NSAP used for local identification, the remaining leading part of which is taken to be the NET(Network Entity Title) The default value is 1, which is conformant to US GOSIP. When an iso address is set in an **ifconfig** command, it is really the NSAP which is being specified.
ptp	- sets the point-to-point flag for the interface (only on GRF HIPPI or ATM interfaces
-ptp	- clears the point-to-point flag
trailers	- requests the use of a ``trailer" link level encapsulation when sending (default). If a network interface supports trailers, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.
-trailers	- disables the use of a ``trailer" link level encapsulation
up	- specifies an interface as up, may be used to enable an interface after an **ifconfig down** If the interface was reset when previously marked down, the hardware is re-initialized.
proxy	- enables Proxy ARP on an interface, allows an interface to respond to ARP requests destined for a host to which the interface has a route
-proxy	- disables Proxy ARP on an interface

iflash
(CLI)

Warning: Perform a **mountf** before you do an **iflash** command.
The system always (and onlys) boots from the boot files resident on internal flash. **iflash** erases boot files on the internal flash device if so directed.

The **iflash** command initializes (formats) specified flash memory.

Permission level: system

Syntax

iflash `[`***option***`]`

Options

-e	- specifies external flash disk, `/dev/wd1a`
-A	- not available
-B	- not available
-i	- specifies internal flash disk, `/dev/wd0a`
-f	- force operation even if device is formatted already
-r *device_name*	- specifies any other device

Examples

Initialize the external flash device in PCMCIA slot A:

```
    iflash -r device_name    or    iflash -e
```

Initialize the internal flash device:

```
    iflash -r device_name    or     iflash -i
```

If you run **iflash -e** against a corrupted external flash, **iflash** should report file system errors and then format the flash device with a "`Formatting...`" message.

If you run **iflash -e** against an external flash that is not corrupted, **iflash** should tell you that there is a file system on the device, and that you need to use the **-f** (force) option to reformat a flash device that contains a valid file system.

list
(CLI)

Lists the field in the current profile. After a profile is read, you can view the contents of that profile by using the **list** command (same as **cd** command).

A second use is to move back (or up) in a profile. The **list ..** command moves you back to the level you were viewing prior to the last **list** or **cd** entered.

See also: **cd**, **get**, **ls**

Permission level: user

Syntax

list [***field-name***] [***field-index***] [...]
list .. [..]

Example

```
super> read dump
DUMP read
super> list
hw-table = <{hippi 20 /var/portcards/grdump 0}{rmb 20 /var/portcards/gr+
dump-vector-table = < {2 fddi"FDDI default dump vectors"<{1"fddi core m+
config-spontaneous = off
keep-count = 0

super> list hw-table
hippi = { hippi 20 /var/portcards/grdump 0 }
rmb = { rmb 20 /var/portcards/grdump 3 }
hssi = { hssi 20 /var/portcards/grdump 7 }
dev1 = { dev1 20 /var/portcards/grdump 9 }
atm-oc3+ = { atm-oc3-v2 20 /var/portcards/grdump 5 }
fddi-v2 = { fddi-v2 20 /var/portcards/grdump 6 }
atm-oc1+ = { atm-oc12-v1 20 /var/portcards/grdump 10 }
etherne+ = { ethernet-v1 20 /var/portcards/grdump 8 }
sonet-v1 = { sonet-v1 20 /var/portcards/grdump 11 }

super> list atm-oc12-v1
media = atm-oc12-v1
config = 20
path = /var/portcards/grdump
vector-index = 10
```

load
(CLI)

Restores (loads) previous configuration script into current use. These files are always located in the `/etc/prof` directory.

Permission level: update

See also: **save**

Syntax

***load** filename*

where ***filename*** is the name of the file in which the configuration script is saved.

Example

Load the previously-saved `system.conf` file:

```
super> load system.conf
SYSTEM read
SYSTEM written
super>
```

If you try to load a file that does not exist, or enter a typing error:

```
super> load superisp.conf
error:  superisp.conf does not exist
super>
```

ls
(CLI)

Displays the contents of the current working profile read into local memory. The **ls** command retrieves the names and contents of fields within profiles without changing the user's location within the tree. **get** is an alias of **ls**.

See also: **cd**, **get**, **list**

Permission level: user

Syntax

ls `[. |` ***profile-type*** `[` ***profile-index*** `] ] [` ***field-name field-index*** `... ]`

Example

When you use **ls** to look at fields in the Dump profile, specify each level of profile:

```
super> read dump
DUMP read

super> ls .
hw-table = <{hippi 20 /var/portcards/grdump 0}{rmb 20 /var/portcards/gr+
dump-vector-table = < { 3 rmb "RMB default dump vectors" <{1 SRAM 262144+
config-spontaneous = off
keep-count = 0
```

Look at the hardware table fields:

```
super> ls . hw
hippi = { hippi 20 /var/portcards/grdump 0 }
rmb = { rmb 20 /var/portcards/grdump 3 }
hssi = { hssi 20 /var/portcards/grdump 7 }
dev1 = { dev1 20 /var/portcards/grdump 9 }
atm-oc3+ = { atm-oc3-v2 20 /var/portcards/grdump 5 }
fddi-v2 = { fddi-v2 20 /var/portcards/grdump 6 }
atm-oc1+ = { atm-oc12-v1 20 /var/portcards/grdump 10 }
etherne+ = { ethernet-v1 20 /var/portcards/grdump 8 }
sonet-v1 = { sonet-v1 20 /var/portcards/grdump 11 }
super>
```

Look at the Ethernet dump fields, specify each level of the profile:

```
super> ls . hw eth
media = ethernet-v1
config = 20
path = /var/portcards/grdump
vector-index = 8
super>
```

man
(CLI+shell)

Returns a man page for the specified command

Permission level: system

Syntax

man ***title*** [**-achw**] [**-C** ***file***] [**-M** ***path***] [**-m** ***path***] [[**-s**] ***section***] ...

title	- name of command man page

Options

-a	- display all of the manual pages for a specified section and name combination
-C	- use the specified file instead of the default configuration file
-c	- copy the manual page to the standard output instead of using **more** to paginate it
-h	- display only the ``SYNOPSIS'' lines of the requested manual pages
-M	- override the list of standard directories **man** searches
-m	- augment the list of standard directories **man**
-s	- restrict the directories to section that man will search
-w	- list the pathnames of the manual pages which **man** would display for the specified section and name combination

Example

Retrieve the **gratm** man page:

```
super> man gratm

GRATM(8)                BSD System Manager's Manual                GRATM(8)
NAME
     gratm - configure GigaRouter ATM cards
                        •
                        •
                        •
```

mem
(CLI)

mem displays how much system RAM is installed on a GRF control board or on an RMS node system's router management board.

```
    super> mem
    System memory 256 Mbytes
```

Permission level: user

Syntax

mem ? or **mem**

mountf
(CLI+shell)

The command mounts a flash device using locks and counts so that multiple processes can take advantage of the mounted device. **mountf** verifies (**fsck**) the flash device before doing the mount. By default, **mountf** mounts a flash device on `/flash`. **mountf** normally mounts the flash device as read-only at the mount point, `/flash.`

Permission level: system

Syntax

mountf [**-d** ***device_descriptor***] [**-i**] [**-e**] [**-A**] [**-B**] [**-m** ***mount_point***] [**-w**]

Options

The **mountf** command supports the following options:

-e	- use the device descriptor for the external flash device: `/dev/wd1a.`
-i	- use the device descriptor for the internal flash device: `/dev/wd0a` (this is the default).
-A	- Use the device descriptor for the external flash device in PCMCIA slot A, `/dev/wd1a` (option not available).
-B	- Use the device descriptor for the external flash device in PCMCIA slot B, `/dev/wd2a` (option not available).
-d *device_descriptor*	- use the specified device descriptor, this option enables another device to be acted upon.
-w	- mount the flash device as writable, the default is to mount the device as read-only.
-m *mount_point*	- loads the device at the specified mount point

netstat
(CLI+shell)

The **netstat** command displays interface routing, protocol, and connection statistics.

See also: **traceroute**, **grrt**

Permission level: system

Syntax

netstat [**-Aan**] [**-f** ***address_family***] [**-M** ***core***] [**-N** ***system***]
netstat [**-dgimnrs**] [**-f** ***address_family***] [**-M** ***core***] [**-N** ***system***]
netstat [**-dn**] [**-I** ***interface***] [**-M** ***core***] [**-N** ***system***] [**-w** ***wait***]
netstat [**-M** ***core***] [**-N** ***system***] [**-p** ***protocol***]

Options

-A	- with the default display, show the address of any protocol control blocks associated with sockets; used for debugging.
-a	- with the default display, show the state of all sockets; normally sockets used by server processes are not shown.
-d	- with either interface display (option **-i** or an interval, as described below), show the number of dropped packets.
-f ***address_family***	- limit statistics or address control block reports to those of the specified address family. The following address families are recognized: **inet**, for AF_INET, **ns**, for AF_NS, **iso**, for AF_ISO, **unix**, for AF_UNIX, and **grit**, for AF_GRIT.
-g	- show information related to multicast (group address) routing. By default, show the IP Multicast virtual-interface and routing tables. If the **-s** option is also present, show multicast routing statistics.
-I ***interface***	- show information about the specified interface; used with a wait interval as described below.
-i	- show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown). If the **-s** option is present more information is specified, but only for a single interface. If the **-a** option is also present, multicast addresses currently in use are shown for each Ethernet interface and for each IP interface address.
-M	- extract values associated with the name list from the specified core instead of the default /dev/kmem.
-m	- show statistics recorded by the memory management routines

-N	- extract the name list from the specified system instead of the default `/bsd`.
-n	- show network addresses as numbers (normally **netstat** interprets addresses and attempts to display them symbolically)
-p ***protocol***	- show statistics about protocol, which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the file `/etc/protocols`.
-s	- show per-protocol statistics. If this option is repeated, counters with a value of zero are suppressed.
-r	- show the routing tables. When **-s** is also present, show routing statistics instead.
-w ***wait***	- show network interface statistics at intervals of ***wait*** seconds

Examples

These netstat commands are useful in the GRF environment, see the *Getting Started* manual:

– **netstat -r -s** prints routing statistics
– **netstat -i -n** shows all configured interfaces
– **netstat -a -n** prints a list of all active connections
– **netstat -r -n** prints the current table of installed routes
– **netstat -s** prints comprehensive statistics for protocols, IP, ICMP, TCP, UDP...

new
(CLI)

The **new** command is used to create a new instance of a profile in local memory. That new instance is not permanent until the profile is written. The profile instance can be a main-level profile or a member of a profile list in a list field.

Permission level: system

Syntax

new ***profile-type*** [***profile-index***]

Example

Create a new main-level User profile for Fred, notice that the default profile is read:

```
super> new user
USER/default read

super> new user bob
USER/fred bob
```

Create a new member of the ports profile list, port 8:

```
super> read card 1
CARD/1 read
super> new ports
ports/0 created

super> new port 8
ports/8 created
super> write
ports/8 written
```

You cannot create a member of a profile list that already exists:

```
super> new port 8
error: profile already exists
```

If you try to create a main-level profile instance that already exists, it will just read the existing profile:

```
super> new user admin
USER/admin read
super>
```

ping
(CLI+shell)

The **ping** command sends echo request packets to elicit a response from a host or gateway.

Permission level: system.

Syntax

ping ***host*** [**-dfLnqRrv**] [**-b** ***sockbuf***] [**-c** ***count***] [**-g** ***groupsize***] [**-I** ***wait***]
[**-i** ***interface***] [**-l** ***preload***] [**-p** ***pattern***] [**-s** ***packetsize***] [**-t** ***ttl***]

host	- network host address

Options

-b ***sockbuf***	- set the kernel socket buffer sizes for both send and receive to size bytes, use if an attempt to send large pings with the **-s** option results in error messages like ``sendto: Message too long."
-c ***count***	- after sending ***count*** echo request packets, **ping** waits a short time---twice the longest response received so far---for outstanding responses, or until all responses are received, whichever is quicker
-d	- set the SO_DEBUG option on the socket being used
-f	- flood ping, outputs packets as fast as they come back or one hundred times per second, whichever is more For every echo request sent, a period ``." is printed, while for every echo reply received, a backspace is printed. This provides a rapid display of how many packets are being dropped. Only the super-user may use this option as it is very hard on a network and should be used with caution.
-g ***groupsize***	- send ***groupsize*** number of packets at a time, the default is to send one echo request each interval.
-I ***wait***	- wait ***wait*** seconds between sending each packet or group of packets, default is to wait one second between each packet or group of packets, incompatible with **-f** option
-i ***interface***	- specify the outgoing ***interface*** to use for multicast packets
-L	- turn off loopback of multicast packets Normally, if there are members in the host group on the outgoing interface, a copy of the multicast packets will be delivered to the local machine.

-l ***preload***	- if ***preload*** is specified, **ping** sends that many packets as fast as possible before falling into its normal mode of behavior
-n	- numeric output only, no attempt is made to lookup symbolic names for host addresses
-p ***pattern***	- you may specify up to 16 ``pad" bytes to fill out the packet you send, useful for diagnosing data-dependent problems in a network  For example, ``**-p ff**" will cause the sent packet to be filled with all ones.
-q	- quiet output, nothing is displayed except the summary lines at startup time and when finished
-R	- record route, includes the RECORD_ROUTE option in the ECHO_REQUEST packet and displays the route buffer on returned packets Note that the IP header is only large enough for nine such routes, many hosts ignore or discard this option.
-r	- bypass the normal routing tables and send directly to a host on an attached network If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it
-s ***packetsize***	- specifies the number of data bytes to be sent, the default is 56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data
-t ttl	- specify the IP time to live **ttl** for multicast packets, the default time to live for multicast is one hop
-v	- verbose output, received ICMP packets other than ECHO_RESPONSE are listed

pinglog
(shell)

The **pinglog** command sends an SNMP trap when a GRF interface or device goes down.

pinglog reads the ping logging file and when a device down message is detected, the message is converted to an SNMP trap and sent to the gateways specified in the SNMP configuration file. The traps generated have the enterprise set to `netstar` (1080) and the general trap-type set to `SNMP_TRAP_ENTERPRISESPECIFIC`. The specific trap-type is set to `grInterfaceDown` (`netstar` enterprise trap 18).

Syntax

pinglog [**-x** ***instance***] [**-g** ***filename***] [**-i** ***timing interval***]

start_pinglog [**-x** ***instance***]

stop_pinglog [**-x** ***instance***]

Options

-x ***instance***	- Identifies the instance number of the process since this module supports multiple instantiations.
-g ***filename***	- A gateway configurations file which specifies the specific trap to be sent. The program allows for on-the-fly change of the gateway configuration file.
-i ***timing interval***	- The timing interval in seconds on how often to check for pinglog and gateway configuration file updates. The default time is set to 5 minutes.

Refer to the **pinglog** man page for more information.

See also **threshpoll**.

port
(grrmb)

This **grrmb** command sets a specified slot as the default slot to be acted upon by **grrmb** commands.

When set, the default slot number remains in effect until changed by the **port** command. If a slot number is specified in any command, the default setting is overridden.

When the **grrmb** interface first starts, the default slot is the control board (or router manager board), 66. After that, the interface retains the last default slot set, and opens the next session with that slot set as default.

Syntax

port *slot number*

Example

To set slot 1 as the default slot, enter:

```
GR 66> port 1
The current port is 1.
GR 01>
```

The prompt is changed to contain the newly specified slot number.

power
(grrmb)

This **grrmb** command displays the on/off status of GRF 1600 power supplies.

Syntax

power

Example

```
GR 15> power
power {1|2} {on|off}
GR 15>
```

pwd (CLI)

Shows the current user location (context) in the profile tree.

The **pwd** command determines the user's current location in the profile tree. The output is similar to a path in a file system. Each level in the tree is separated by forward slashes (/). If a profile is indexed, that is, if it is a member of a list, the index will follow the profile name. **whereami** is an alias for **pwd**.

Permission level: user.

Syntax

pwd

Example

whereami and **pwd** are used interchangeably:

```
super> read card 2
CARD/2 read
super> whereami
CARD 2

super>cd ports 1
port_num = 1
hippi = { 1 32 0 999999 4 3 5 300 0 10 10 03:00:0f:c0 disabled 0 0 +
fddi = { single }
sonet = { "" "" 1 sonet internal-oscillator 0 }
hssi = { False 16-bit }
ether = { autonegotiate disabled }

super>pwd
CARD 2/ports 1
super>
```

pwrfaild
(shell)

This command supports an optional uninterruptible power source (UPS) to which the RMS node can be connected. The software is designed to work with a specific cable that connects the RMS node to a specific APC UPS model.

When the UPS signals that main power has been lost**, pwrfaild** shuts the system software down gracefully.   The RMS node must be connected to the UPS via an APC SU700RM Control Cable (part number CABLE0053).

Warning: A standard RS-232 cable cannot be used to connect the RMS node to the UPS.

This type of cable will not be able to read power state signals from the UPS and, further, such a cable could damage serial ports on both the UPS and the RMS node.

pwrfaild has been tested with the APC Smart-UPS 700 Uninterruptible Power Source. Other APC models with serial port signalling identical to this model should work with this software.

If you already have another type of UPS, compare the serial port wiring information on the next page to decide if that UPS would be compatible with this software.

To configure **pwrfaild** so that it is started when the RMS boots, uncomment or add this code at the end of `/etc/grstart`:

```
if [-x /usr/nbin/pwrfaild ] ; then
     /usr/nbin/pwrfaild
fi
```

Default tty port

The default tty port used by **pwrfaild** is `/dev/tty01`. This port is labelled “COM 2” and is on the back of the RMS node. The RS-232 (`/dev/tty00`) port on the front of the RMS node is reserved for the system console. This port can be the control connection to the UPS. See the *Optional try port* section.

Older RMS systems

A previous generation of RMS systems are based on PCs that have only a single serial port (`/dev/tty00`). To invoke **pwrfaild** with this type of system, modify `/etc/grstart` with the `-f /dev/tty00` argument:

```
if [ -x /usr/nbin/pwrfaild ] ; then
        /usr/nbin/pwrfaild -f /dev/tty00
fi
```

Customers upgrading from earlier systems who use **pwrfaild** must take one of two actions depending on their type of RMS hardware:

Customers with Compaq Presario RMS systems must change `/etc/rc.local`, and use the **-f** option to specify that **pwrfaild** should continue to use serial port tty00, as follows:

```
/usr/nbin/pwrfaild -f /dev/tty00
```

Syntax

pwrfaild [**-D**] [**-d**] [**-f** **/dev/ttyXX**] [**-t** ***seconds***]

Options

-d	- turns on debugging
-D	- prevents **pwrfaild** from detaching from its controlling tty (so it can be run under a debugger)
-**f** **/dev/ttyXX**	- specifies *XX* tty device to use for the connection to the UPS (default tty used by **pwrfaild** is `/dev/tty01`)
-t ***seconds***	- sets the number of seconds that the RMS can draw power from the UPS before the RMS begins the shutdown sequence The default for this parameter is 30 seconds. If main power is lost, but not restored after the number of seconds specified in this parameter, the RMS will shut down. If main power is restored within this time, the RMS does not shut down. The shutdown timer is reset, so that the next time power fails, the full value of this parameter must elapse before the RMS shutdown sequence commences.

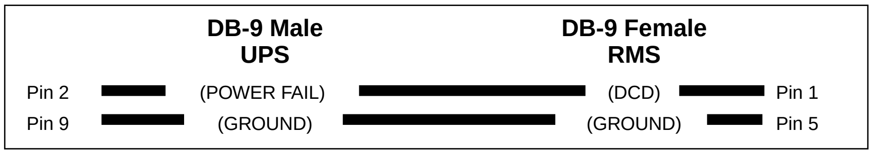
Cable signalling specifications

The following DB-9 cable design is used to connect an APC Back-UPS 700 to the GRF RMS system. Customers should obtain this cable directly from Ascend to insure proper wiring and to avoid damage to serial ports on either the RMS or the APC UPS.

This wiring diagram is provided for informational purposes only.

– Pin 2 on the UPS (normally low) should be wired to DCD (CAR) on the RMS serial port (pin 1).
– Pin 9 on the UPS should be connected to ground on the RMS (pin 5).
– The rest of the wires, if any, should be cut. Only 24-gauge solid copper wire should be used.

The resulting cable should look like this diagram:



quit
(CLI)

Terminates current CLI session. If the session originated from a remote device, an associated connection is terminated (telnet or modem connection). If the session is on a local console and system-wide authentication is in use, the login prompt is issued.

Permission level: user

Syntax

quit

read
(CLI)

Before you are able to look at and/or change any data, you must first 'read' the profile in which the data is stored into local memory. You can only work with one profile at a time. Once you read another profile, the profile that was previously in local memory is now replaced by the new profile. Once the profile is in local memory, you may look at and/or change the data in the profile. If you do make a change, you need to save that change to make it permanent. Some profiles are read-only. This is noted in the response to the read command. Once a profile is read, you will receive a response indicating that the read was successful.

Permission level: user

Syntax

read ***profile-type*** [***profile-index***]

Example

Read a one-of-a-kind profile:

```
super> read system
SYSTEM read
```

Read a many-of-a-kind profile:

```
super> read user default
USER/default read
```

route
(CLI+shell)

Adds or deletes static routes manually if dynamic routing is not running.

Permission level: system

Syntax

route [**-nqv**] ***command*** [[***modifiers***] ***args***]

command The **route** utility provides six commands:

add	- add a route
flush	- remove all routes
delete	- delete a specific route
change	- change aspects of a route (such as its gateway)
get	- lookup and display the route for a destination
monitor	- continuously report any changes to the routing information base, routing lookup misses, or suspected network partitionings

The **monitor** command has the syntax: `route [-n] monitor`

The **flush** command has the syntax: `route [-n] flush [family]`

If the **flush** command is specified, **route** will ``flush'' the routing tables of all gateway entries. When the address family is specified by any of the -osi, -xns, or -inet modifiers, only routes having destinations with addresses in the delineated family will be deleted.

The other commands have the following syntax:

route [**-n**] ***command*** [**-net** | **-host**] ***destination gateway***

where ***destination*** is the destination host or network, ***gateway*** is the next-hop intermediary via which packets should be routed. Routes to a particular host may be distinguished from those to a network by interpreting the Internet address specified as the destination argument. The optional modifiers **-net** and **-host** force the destination to be interpreted as a network or a host, respectively. Otherwise, if the destination has a ``local address part'' of INADDR_ANY , or if the destination is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host.

Options

-n	- bypasses attempts to print host and network names symbolically when reporting actions
-v	- enable verbose mode, print additional details
-q	- suppress all output

save (CLI)

The **save** command saves the current profile configuration in a script form to permanent storage. This file can then be loaded at a later date to restore the previous configuration.

Permission level: update

Syntax

Syntax to save the configuration to a file:

save [**-a**] [**-m**] ***filename*** [***profile-type*** [***profile-index***]]

Syntax to write the configuration only to the screen, not to a file:

save [**-a**] [**-m**] **console** [***profile-type*** [***profile-index***]]

If a profile-type is not specified, all savable profiles will be saved. If a profile-type is specified, but a profile-index is not specified (and it is a multiple-instance profile), all profiles of that type will be saved.

If the **-a** option is specified, all fields will be explicitly saved. Otherwise, only those fields whose contents differ from the default values will be saved.

If **-m** is specified, all fields will be saved by their field numbers rather than their field names.

If the current user does not have password accessiblity, a message will appear warning the user not to save any profiles that contain passwords. This is done because all passwords will be written as strings of stars.

All files will be saved in the `/etc/prof` directory. If the specified profile already exists, a message will appear warning the user that this file already exists and asking the user if s/he wants to overwrite this file.

Example

Save all savable profiles to a specified file

```
super> save all.conf
super>
```

Save all user profiles to a specified file:

```
super> save user.conf user
super>
```

Save the User admin profile to a specified file:

```
super> save admin.conf user admin
super>
```

Saving to a file that already exists, resulting warning:

```
super> save all.conf card
```

```
WARNING: all.conf already exists.  If you choose to save to this file,
all configuration information that now exists in all.conf will be over-
written. Continue? [y/n] n
save aborted
super>
```

Save a user profile when you do not have password access:

```
super> save default.conf user default

WARNING: the current user has insufficient rights to view password
fields.  A configuration saved under this circumstance should not be used
to restore profiles containing passwords.

Save anyway? [y/n] n
super>
```

set
(CLI)

The **set** command is used to modify fields in the last profile read. Modifications do not take effect until the profile is written, needs write. A second function returns help text about a field.

Permission level: system

Syntax

The first format is used to change a field, the field-name must match a name in the last profile read. When changing a field, the field-value is everything between the white space following the = and the end of the line.

set *field-name* = ***field-value***

This usage returns help on the type of values to which the field can be set.

set *field-name* ?

Example

Read the User profile to create a new copy of the default:

```
super> read user default
USER/default read
super> list
```

Now set the fields to be changed:

```
super> set name = operator
super> set password = mypasswd
super> set active-enabled = yes
super> set allow-system = yes
super> set allow-update = yes
super> set prompt = operator
```

Check the new values?

```
super> list
name* = operator
password = mypasswd
auth-method = { PASSWD { "" 1645 udp "" } { 5500 udp 5510 tcp +
active-enabled = yes
allow-system = yes
allow-update = yes
allow-password = no
allow-debug = no
prompt = operator=>
log-display-level = none
super>
```

The field-value is everything between the white space following the = and the end of the line. In this example, `set prompt` is the field, `operator` is the field-value:

```
set prompt = operator
```

setver
(CLI+shell)

The **setver** command sets the next version of software to run on the next system boot. **setver** mounts the flash device specified and verifies that the software version on the flash device has the correct pieces needed to boot.

setver checks to see if the release tar file exists, then checks that the boot, bsd, and startup scripts exist. Finally, it checks that the memory file system and the configuration files exist.

Once files are verified, **setver** moves, extracts, or links on the flash device all the files that it has verified.

Permission level: system

See also: **getver**, **grfins**

Syntax

setver ***revision*** [**-e**] [**-A**] [**-B**] [**-i**] [**-p** ***path***] [**-d** ***device_descriptor***]

where:

revision	- specifies the software release and version against which to perform the command

Options

The **setver** command supports the following options:

-e	- use the device descriptor for the external flash device: `/dev/wd1a`
-i	- use the device descriptor for the internal flash device: `/dev/wd0a` (this is the default)
-A	- (option not available) use the device descriptor for the external flash device in PCMCIA slot A, `/dev/wd1a`
-B	- (option not available) use the device descriptor for the external flash device in PCMCIA slot B, `/dev/wd2a`
-d ***device_descriptor***	- use the specified device descriptor, this option enables another device to be acted upon
-p ***path***	- use specified path, and do not mount (used when flash device is already mounted

This option is in the form *release,version*. For example, `1.4.x,default` or `1.4.x,atmtestB`.
The currently-running version is assigned by **flashcmd** to be `default`.

sh (CLI)

The **sh** command creates a UNIX shell in the CLI. Always use **exit** or **quit** to return to the CLI prompt.

You can create one UNIX shell per CLI session, multiple shells and sessions do not nest within each other. You receive a warning message if you attempt to start a second CLI from the UNIX shell.

Permission level: user

Syntax

sh

Example

```
super> sh
#
```

shutdown
(CLI+shell)

Halts, reboots, shuts down the operating system. When the GRF is shut down, media card operations stop abruptly. Connections are broken and enroute packets are lost.

Refer to the **shutdown** man page for more information.

Permission level: system

Syntax

shutdown [-] [**-fhikrn**] **time**

Options

-f	- **shutdown** arranges, in the manner of **fastboot**, for the file systems not to be checked on reboot.
-h	- the system is halted at the specified time when **shutdown** executes **halt**
-i	- the **-i** option is only available on the GRF system.

This option is an override flag to ignore file system check. The GRF system has a built-in safety feature where it prevents the administrator from accidentally rebooting the system without saving the configuration files in `/etc`. It determines this by running the **grwrite** command.

If the **grwrite** command exits with a non-zero exit status, then a warning message is printed and the shutdown operation is aborted. The **-i** option is used to override this safety feature. This option is not applicable on a non GRF system.

-r	- **shutdown** execs **reboot** at the specified time
time	- ***time*** is the time at which shutdown will bring the system down, use “now” to indicate an immediate shutdown

Example

Log in as root and prepare to power off the GRF system:

```
super> shutdown -h now
```

Log in as root and prepare to reboot the GRF system:

```
# shutdown -r now
```

temp
(grrmb)

The **temp** command reads temperature sensors mounted on the control board and returns the chassis’s internal temperature.

Syntax

temp

Example

To print its contents and then clear the trace buffer, enter:

```
GR 66> temp
GR 66>  temp

                     LT     HT    TEMP
                  ---------------------
Measured Junction     53 C   58 C   39 C
Calc. Ext. Ambient    47 C   52 C   33 C

GR 66>
```

LT	- lower temperature threshold (start of software shutdown begins)
HT	- high temperature threshold (hardware shutdown - immediate)
TEMP	- current operating temperature
Measured Junction	- temperature on board surface
Calculated External Ambient	- estimated temperature of air surrounding the chassis

threshpoll
(shell)

The **threshpoll** command sends an SNMP trap if a threshold condition is detected for a media card operation.

Currently, thresholding is only supported for the following objects:

- ifInOctets
- ifOutOctets
- ifInUcastPkts
- ifOutUcastPkts
- ifInErrors
- ifOutErrors
- ifInDiscards
- ifOutDiscards

For all of these objects, thresholding is done on the delta of the current value and the previous value.

A poll group file determines the object identifiers to poll and the trap-type to send when the threshold condition is met. A thresholding file determines the thresholding operation and the value associated with each poll's object identifiers. A device setup file which determines the list of devices to poll and the poll groups associated with each device. In addition, this file also indicates which poll group to threshold on a per device, per poll group basis.

The specified values are thresholded against the values in the thresholding configuration file. If the thresholding condition is met, an SNMP trap is output to the gateway specified in the SNMP configuration file (`/etc/snmpd.conf`). If the log file environment variable is set, then values will be logged to a `/var/log` file.

Syntax

threshpoll -x ***instance*** [**-i** ***timing interval***]

start_threshpoll -x ***instance***

stop_threshpoll -x ***instance***

Option

-x ***instance***	- Identifies the instance number of the process since this module supports multiple instantiations.
-i ***timing interval***	- The timing interval in seconds on how often to poll the defined interfaces. The default time is set to 5 minutes.

Refer to the **threshpoll** man page for more information.

traceroute
(CLI+shell)

Prints the packet route to a specified destination host/network.

See also: **netstat**, **grrt**

Permission level: system

Syntax

traceroute [-m *max_ttl*] [-n] [-p *port*] [-q *nqueries*] [-r] [-s *src_addr*]

[-t *tos*] [-w *waittime*] *host* [*packetsize*]

host	- destination host name or IP number

Options

-m *max_ttl*	- set the max time-to-live (max number of hops) used in outgoing probe packets The default is 30 hops (the same default used for TCP connections).
-n	- print hop addresses numerically rather than symbolically and numerically (saves a nameserver address-to-name lookup for each gateway found on the path)
-p *port*	- set the base UDP port number used in probes (default is 33434)
-q *nqueries*	- set the number of probes per ``ttl'' to nqueries (default is three probes).
-r	- bypass the normal routing tables and send directly to a host on an attached network If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped by routed).
-s *src_addr*	- use the following IP address (which must be given as an IP number, not a hostname) as the source address in outgoing probe packets On hosts with more than one IP address, this option can be used to force the source address to be something other than the IP address of the interface the probe packet is sent on. If the IP address is not one of this machine's interface addresses, an error is returned and nothing is sent.
-t *tos*	- set the type-of-service in probe packets to the following value, default is zero

	The value must be a decimal integer in the range 0 to 255. This option can be used to see if different types-of-service result in different paths.
-v	- verbose output Received ICMP packets other than TIME_EXCEEDED and UNREACHABLEs are listed.
-w *waittime*	- set the***waittime*** (in seconds) to wait for a response to a probe, default is 3 seconds
	- specify the packet size in bytes after the destination host name ***host*** The default probe datagram length is 38 bytes.

umountf
(CLI+shell)

The **umountf** command unmounts a flash device that was previously mounted using the **mountf** command.

Permission level: system

Syntax

umountf [**-d** ***device_descriptor***] [**-e**] [**-i**] [**-A**] [**-B**]

Options

The **umountf** command supports the following options:

-e	- use the device descriptor for the external flash device: `/dev/wd1a`
-i	- use the device descriptor for the internal flash device: `/dev/wd0a` (this is the default)
-A	- (option not available) use the device descriptor for the external flash device in PCMCIA slot A, `/dev/wd1a`
-B	- (option not available) use the device descriptor for the external flash device in PCMCIA slot B, `/dev/wd2a`
-d ***device_descriptor***	- use the specified device descriptor, this option enables another device to be acted upon

vpurge
(CLI+shell)

The **vpurge** command removes a configuration from a flash device. **vpurge** removes the files **snapshot** puts on a flash drive. This includes the main `TAR.gz`, `root.dd.gz`, and the `/etc` release directory.

Permission level: system

Syntax

vpurge --release=*release*,*version* *options*

Options

--force	- forces **vpurge** to remove the files even when they are in use or are set to run at next boot
	
	Warning messages are returned when **--force** is specified:
	`“fatal: is the next boot release”`
	`“fatal: is the running release”`

--release	- sets the release version in the form *x.x.x,<version>*, for example, `1.4.x,default`
--target=i **--target=e**	- specifies the target device, currently internal (**i**) or external (**e**)
--verbose	- prints out more information as to what is going on
--help	- shows a short help message

whatami
(CLI)

Tells you if the system is RMS node-based (`IRMS`) or has the new control board (`CB`).
Permission level: system

Syntax

whatami

Example

```
super> whatami
CB
super> whatami
IRMS
```

whereami
(CLI)

Shows the current user location (context) in the profile tree.

The **whereami** command determines the user's current location in the profile tree. The output is similar to a path in a file system. Each level in the tree is separated by forward slashes (/). If a profile is indexed, that is, if it is a member of a list, the index will follow the profile name. **pwd** is an alias for **whereami**.

Permission level: user.

Syntax

whereami

Example

whereami and **pwd** are used interchangeably:

```
super> read card 2

CARD/2 read
super> whereami
CARD 2

super> cd ports 1
port_num = 1
hippi = { 1 32 0 999999 4 3 5 300 0 10 10 03:00:0f:c0 disabled 0 0 +
fddi = { single }
sonet = { "" "" 1 sonet internal-oscillator 0 }
hssi = { False 16-bit }
ether = { autonegotiate disabled }

super> pwd
CARD 2/ports 1
super>
```

whoami
(CLI)

The whoami command returns the user profile name associated with this session.

Permission level: user

Syntax

whoami

Example

```
super> read card 3
CARD/3 read
card-num* = 3
media-type = fddi-v2
debug-level = 0
hssi-frame-protocol = Frame-Relay
sonet-frame-protocol = PPP
ether-verbose = 0
ports = < { 0{dual off}{ "" "" 1 sonet internal-oscillator 0}{0 16-bit }+
load = { 0 < > 1 0 0 }
dump = { 0 < > 0 }
config = { 0 1 1 4 0 0 }
icmp-throttling = { 10 10 2147483647 10 10 10 }

super> list . du
config = 0
hw-table = < >
config-spontaneous = 0

super> who
bob
super>
```

write
(CLI)

The **write** command validates the profile and stores a local profile into memory. **write** is used to apply changes you have made to a field value, and is part of the process to create a new instance of a User or Card profile.

A profile-type or profile-index is not required for the **write** command as the current profile is always the one written.

Permission level: update

Syntax

write

Example

Use write to store a new version of a User profile:

```
super> dir user
92  01/31/97 10:16:08  default
103  01/31/97 10:16:09  admin
106  01/31/97 10:16:10  super
99  02/03/97 15:27:00  bob

super> read user bob
USER/bob read

super> set name = bob2

super> write
USER/bob2 written

super> dir user
 92  01/31/97 10:16:08  default
103  01/31/97 10:16:09  admin
106  01/31/97 10:16:10  super
 99  02/03/97 15:27:00  bob
100  02/03/97 16:00:00  bob2
```

GateD Configuration Statements

2

This release of the Ascend Embedded Operating System supports version 3.5b3 of GateD.

Chapter 2 contains the following GateD configuration topics and statements:

Introduction

As a contributing member of the GateD Consortium, Ascend’s High Performance Networking Division is developing additions to portions of GateD and particularly to BGP:

- Configurable Export-Best-BGP (CEEB), send best BGP route when route from another protocol is active
- Asynchronous Multi-level Next Hop Resolution, nexthopself within IBGP
- AS path truncate variable added
- BGP enhancements: policy per peer, peer group with separate policy per peer; send original BGP next hop.
- new AS path length algorithm
- increased number of GateD adjacencies / peering sessions
- IS-IS
- new monitoring tool, GateD State Monitor
- BGP confederations
- BGP Multi Exit Discriminator (MED)
- communities
- route reflection
- route filtering
- routing arbiter interaction
- route preference biasing
- stability support for BGP-OSPF interaction
- a weighted route dampening statement

Syntax

The GateD configuration file consists of a sequence of statements terminated by a semi-colon (`;').

Statements are composed of tokens separated by white space, which can be any combination of blanks, tabs, and new lines. This structure simplifies identification of the parts of the configuration associated with each other and with specific protocols.

Comments may be specified in either of two forms:

- One form begins with a pound sign (`#') and runs to the end of the line.
- The other form, C style, starts with a `/*' and continues until it reaches `*/'.
-

Syntax description conventions

Keywords and special characters that the parser expects exactly are shown in **bold** font. Variable parameters are shown in *italic* font. Square brackets ([and]) are used to show optional keywords and parameters. The vertical bar (|) separates optional parameters. Parentheses ((and)) group keywords and parameters when necessary.

For example, in the syntax description:

[**backbone** | (**area** *area*)]

the square brackets say that either parameter is optional. Keywords are **backbone** and **area**. The vertical bar indicates that either ``**backbone**'' or ``**area** *area*'' may be specified. Since *area* is in *italic* font, it is a parameter that needs to be provided.

Statement grouping

The configuration statements and the order in which these statements appear divide `/etc/gated.conf` into options statements, interface statements, definition statements, protocol statements, static statements, control statements, and aggregate statements.

Entering a statement out of order causes an error when parsing the configuration file.

Two types of statements do not fit in these categories: %directive statements and %trace statements. These statements provide instructions to the parser and control tracing from the configuration file. They do not relate to the configuration of any protocol and may occur anywhere in the `/etc/gated.conf` file.

Statement summary

A summary table of the configuration statements follows this section. The table lists

each GateD configuration statement by name, identifies the statement type, and provides a short synopsis of the command's function. More detailed definitions and descriptions of each of the eight classes of GateD statements follow in separate sections.

Statement summary

Table 2-1 lists each statement name and type, and gives a synopsis of the statement's function.

Table 2-1. Summary of GateD configuration statements

Statement	**Type**	**Function**
%directory	(directive)	Sets the directory for include files.
%include	(directive)	Includes a file into `gated.conf`.
traceoptions	(trace)	Specifies which events are traced.
options	(definition)	Defines GateD options.
interfaces	(definition)	Defines GateD interfaces.
autonomoussystem	(definition)	Defines the AS number.
routerid	(definition)	Defines the originating router (BGP, OSPF).
martians	(definition)	Defines invalid destination addresses.
rip	(protocol)	Enables RIP protocol.
hello	(protocol)	Enables HELLO protocol.
isis	(protocol)	Enables ISIS protocol.
kernel	(protocol)	Configures kernel interface options.
ospf	(protocol)	Enables OSPF protocol.
egp	(protocol)	Enables EGP protocol.
bgp	(protocol)	Enables BGP protocol.
redirect	(protocol)	Configures the processing of ICMP redirects.
weighted route dampening	(protocol)	Minimizes effects of route flapping
icmp	(protocol)	Configures the processing of general ICMP packets.
snmp	(protocol)	Enables reporting to SNMP.
static	(static)	Defines static routes.
import	(control)	Defines which routes to import.
export	(control)	Defines which routes to export.
aggregate	(control)	Defines which routes to aggregate.
generate	(control)	Defines which routes to generate.

Protocol-precedence and preference

Protocol-precendence is the value GateD uses to order precedence of routes from one protocol over another. Preference is the value GateD uses to order the preference of routes within one protocol. Precedence and preference can be set in the GateD configuration files in several different configuration statements. Preference can be set based on network interface over another, from one protocol over another, or from one remote gateway over another.

Preference may not be used to control the selection of routes within an **igp**, this is accomplished automatically by the protocol based on metric. Preference may be used to select routes from the same **egp** learned from different peers or autonomous systems. Each route has only one preference value associated with it, even though preference can be set at many places in the configuration file. Simply, the last or most specific preference value set for a route is the value used. (See the Glossary: Preference or Glossary: Precedence entries, page 2-110.) The preference value is an arbitrarily assigned value used to determine the order of routes to the same destination in a single routing database. The active route is chosen by the lowest preference value. The range of preference values is 0 - 65536.

Note: It is not recommended that precedence ever be set. It is recommended that precedence always be used with the given defaults. One can create routing loops, or seemingly indeterminant routing may occur. It is only meant for use with the upmost care and knowledge. Last, note that precedence can be set in all places that preference can be set.

Selecting a route

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors. GateD uses the following criteria, in the order given, to select the best path. If routes are equal at a given point in the selection process, then the next criterion is applied to break the tie.

- Configured Policy - The route with the best (numerically smallest) preference, as determined by the policy defined in `gated.conf`.
- Local_Pref - The route with the highest local preference. Local_Pref is used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route.Local_Pref can be set using the option 'localpref' on the Import or Export statement.
- Shortest AS Path - The route whose NLRI specifies the smallest set of destinations.
- Origin **igp** < **egp** < Incomplete - The route with an AS path origin of **igp** is preferred. Next in preference is the route with AS path origin of **egp**. Least preferred is an AS path that is incomplete.
- MED (if not ignored) - The route with the best Multi-Exit Discriminator is preferred.
- Shortest **igp** distance - If both routes are from the same protocol and AS, the route with the lower metric is preferred.
- Source **igp** < **ebgp** < **ibgp** - Prefer first the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.
- Lowest Router ID - The route whose next hop is the lowest numeric IP address.

Assigning protocol-precedences

A default precedence is assigned to each source from which GateD receives routes. Precedence values range from 0 to 255 with the lowest number indicating the most preferred route.

Table 2-2 summarizes the default preference values for routes learned in various ways.

The table lists the statements (some of these are clauses within statements) that set preference, and shows the types of routes to which each statement applies. The default preference for each type of route is listed, and the table notes preference precedence between protocols. The narrower the scope of the statement, the higher precedence its preference value is given, but the smaller the set of routes it affects.

Table 2-2. Precedence values

Precedence of:	Defined by statement:	Default value:
direct connected networks	**interface**	`0`
OSPF routes	`ospf`	`10`
IS-IS level 1 routes	**isis level 1**	`15`
IS-IS level 2 routes	**isis level 2**	`18`
internally generated default	**gendefault**	`20`
redirects	**redirect**	`30`
routes learned via route socket	**kernel**	`40`
static routes from config	**static**	`60`
ANS SPF (SLSP) routes	**slsp**	`70`
HELLO routes	**hello**	`90`
RIP routes	**rip**	`100`
point-to-point interface		`110`
routes to interfaces that are down	**interfaces**	`120`
aggregate/generate routes	**aggregate/generate**	`130`
OSPF AS external routes	**ospf**	`150`
BGP routes	**bgp**	`170`
EGP	**egp**	`200`

Sample preference specifications

```
interfaces {
      interface 138.66.12.2 preference 10 ;
   } ;
   rip yes {
       preference 90 ;
   } ;
   import proto rip gateway 138.66.12.1 preference 75 ;
```

In these statements, the preference applicable to routes learned via RIP from gateway 138.66.12.1 is 75. The last preference applicable to routes learned via RIP from gateway 128.66.12.1 is defined in the Accept statement. The preference applicable to other RIP routes is found in the RIP statement. The preference set on the Interface statement applies only to the route to that interface.

Trace statements and global options

Trace statements control tracing options.

GateD tracing options can be configured at many levels. Tracing options include the file specifications, control options, and global and protocol-specific tracing options. Unless overridden, tracing options from the next higher level are inherited by lower levels. For example, BGP peer tracing options are inherited from BGP group tracing options, which are inherited from global BGP tracing options, which are inherited from global GateD tracing options. At each level, tracing specifications override the inherited options.

Global tracing options

There are two types of global options: those which only affect global operations and those which have potential significance to protocols.

Global significance only

The trace flags that only have global significance are:

parse
Trace the lexical analyzer and parser, mostly used by GateD developers for debugging.

adv
Trace the allocation of and freeing of policy blocks, mostly used by GateD developers for debugging.

symbols
Used to trace symbols read from the kernel at start-up.
The only useful way to specify this level of tracing is via the `-t` option on the command line, since the symbols are read from the kernel before parsing the configuration file.

iflist
Used to trace the reading of the kernel interface list.
It is useful to specify this with the `-t` option on the command line since the first interface scan is done before reading the configuration file.

Protocol significance

The options flags that have potential significance to protocols are:

all
Turn on all of the following.

general
A shorthand notation used for for specifying both `normal` and `route.`

state
Trace state machine transitions in the protocols.

normal

Trace normal protocol occurrences, abnormal protocol occurrences are always traced.

policy

Trace application of protocol- and user-specified policy to routes being imported and exported.

task

Trace system interface and processing associated with this protocol or peer .

timer

Trace timer usage by this protocol or peer.

route

Trace routing table changes for routes installed by this protocol or peer.

Note: Not all of the above options apply to all of the protocols. In some cases their use does not make sense (for instance, RIP does not have a state machine), and in some instances the requested tracing has not been implemented (such as RIP support of the `policy` option).

Note: It is not currently possible to specify packet tracing from the command line. This is because a global option for packet tracing would potentially create too much output.

When protocols inherit their tracing options from the global tracing options, tracing levels that do not make sense (such as `parse`, `adv` and packet tracing options) are masked out.

Global tracing statements have an immediate effect, especially parsing options that affect the parsing of the configuration file. Tracing values inherited by protocols specified in the configuration file are initially inherited from the global options that are currently in effect as the protocol configuration entries are parsed, unless they are overridden by more specific options. After the configuration file is read, tracing options that were not explicitly specified are inherited from the global options in effect at the end of the configuration file.

Packet tracing

Tracing of packets is very flexible. For any given protocol there are one or more options for tracing packets. All protocols allow use of the **packets** keyword for tracing *all* packets sent and received by the protocol. Most protocols have other options for limiting tracing to a useful subset of packet types. These tracing options can be further controlled with the following modifiers:

detail

detail must be specified before **send** or **recv**. Normally packets are traced in a terse form of one or two lines. When **detail** is specified, a more verbose format is used to provide additional detail on the contents of the packet.

send or **recv**

These options limit the tracing to packets sent or received. Without these options, both sent and received packets will be traced.

Note: **detail**, if specified, must be before **send** or **recv**. If a protocol allows for several different types of packet tracing, modifiers may be applied to each individual type. But be aware that within one tracing specification the trace flags are summed up, so specifying **detail** packets will turn on full tracing for all packets.

Traceoptions syntax

> **traceoptions** ["*trace_file*" [**replace**] [**size** *size*[**k**|**m**] **files** *files*]]
>   [*control_options*] *trace_options* [**except** *trace_options*] ;
>
> **traceoptions** none ;

trace_file

Specifies the file to receive tracing information. If this file name does not begin with a slash (/), the directory where GateD was started is prepended to the name.

replace

Tracing should start by replacing an existing file. The default is to append to an existing file.

size *size* [**k**|**m**] **files** *files*

Limit the maximum size of the trace file to the specified size (minimum 10k). When the trace file reaches the specified size, it is renamed to `file.0`, then `file.1`, `file.2` up to the maximum number of files (minimum specification is 2).

control_options

Specifies options that control the appearance of tracing. Valid values are:

nostamp

Specifies that a timestamp should not be prepended to all trace lines.

trace_options
except *trace_options*

Used to enable a broad class of tracing and then disable more specific options.

none

Specifies that all tracing should be turned off for this protocol or peer.\

Directive statement

Directive statements provide direction to the GateD configuration language parser about included files and the directories in which these files reside.

Directive statements are immediately acted upon by the parser. Other statements terminate with a semi-colon (;), but directive statements terminate with a newline. The two directive statements are:

%directory `"directory"`

Defines the directory where the include files are stored.
When it is used, GateD looks in the directory identified by pathname for any included files that do not have a fully qualified filename, i.e. do not begin with "/". This statement does not actually change the current directory, it just specifies the prefix applied to included file names.

%include `"filename"`

Identifies an include file.
The contents of the file is *included* in the `gated.conf` file at the point in the `gated.conf` file where the `%include` directive is encountered. If the filename is not fully qualified, i.e. does not begin with "/", it is considered to be relative to the directory defined in the `%directory` directive. The `%include` directive statement causes the specified file to be parsed completely before resuming with this file. Nesting up to ten levels is supported. The maximum nesting level may be increased by changing the definition of **FI_MAX** in `parse.h`.

In a complex environment, segmenting a large configuration into smaller, more easily understood segments might be helpful, but one of the great advantages of GateD is that it combines the configuration of several different routing protocols into a single file. Segmenting a small file unnecessarily complicates routing configurations.

Note: These statements must begin in the first character column of the `/etc/gated.conf` file. In other words, there can be no tabs or spaces before the %.

Options statement

Options statements allow specification of some global options. If used, **options** must appear before any other type of configuration statement in the `gated.conf` file.

The options statement syntax is:

```
options
    [ nosend ]
    [ noresolv ]
    [ gendefault [ precedence precedence ] [ gateway gateway] ]
    [ syslog [ upto ] log_level ]
    [ mark time ]
    ;
```

The options list can contain one or more of the following options:

gendefault [**precedence** *precedence*] [**gateway** *gateway*]

When **gendefault** is enabled, when a BGP or EGP neighbor is up it causes the creation of a default route with the special protocol `default`. This can be disabled per BGP/EGP group with the `nogendefault` option. By default, this route has a precedence of 20. This route is normally not installed in the kernel forwarding table, it is only present so it can be announced to other protocols. If a gateway is specified, the default route will be installed in the kernel forwarding table with a next hop of the listed gateway.

Note: the use of the more general option is preferred to the use of this **gendefault** option. The **gendefault** option may go away in future releases. See the section on Route Aggregation for more information on the Generate statement (page 2-100).

nosend

Do not send any packets. This option makes it possible to run GateD on a live network to test protocol interactions without actually participating in the routing protocols. The packet traces in the GateD log can be examined to verify that GateD is functioning properly. This is most useful for RIP and HELLO, and possibly the SMUX SNMP interface. This option does not yet apply to BGP and is less than useful with EGP and OSPF.

noresolv

By default, GateD will try to resolve symbolic names into IP addresses by using the `gethostbyname`() and `getnetbyname`() library calls. These calls usually use the Domain Name System (DNS) instead of the hosts local host and network tables. If there is insufficient routing information to send DNS queries, GateD will deadlock during start-up. This option can be used to prevent these calls, symbolic names will result in configuration file errors.

syslog [**upto**] *log_level*

Controls the amount of data GateD logs via **syslog** on systems where `setlogmask`() is supported. The available logging levels and other terminology are as defined in the `setlogmask`(3) man page. The default is equivalent to `syslog upto` info.

mark `time`

Specifying this option causes GateD to output a message to the trace log at the specified interval. This can be used as one method of determining if GateD is still running.

Interfaces statement

Syntax

```
interfaces {
    options
        [ strictinterfaces ]
        [ scaninterval time ]
        ;
    interface interface_list
        [ precedence precedence ]
        [ down precedence precedence ]
        [ passive ]
        [ simplex ]
        [ reject ]
        [ blackhole ]
        ;
    define address
        [ broadcast address ] | [ pointtopoint address ]
        [ netmask mask ]
        [ multicast ]
        ;
} ;
```

An interface is the connection between a router and one of its attached networks. A physical interface may be specified by interface name, by IP address, or by domain name (unless the network is an un-numbered point-to-point network).

Multiple levels of reference in the configuration language allow identification of interfaces using wildcard, interface type name, or delete word address. Be careful with the use of interface names as future Unix operating systems may allow more than one address per interface. The `interface_list` is a list of one or more interface names including wildcard names (names without a number) and names which may specify more than one interface or address, or the token `all` for all interfaces.

options

Allows configuration of some global options related to interfaces. These are:

strictinterfaces

Indicates that it is a fatal error to reference an interface in the configuration file that is not present when GateD is started and not listed in a **define** statement. Without this option, a warning message will be issued but GateD will continue.

scaninterval `time`

Specifies how often GateD scans the kernel interface list for changes. The default is every 15 seconds on most systems, and 60 seconds on systems that pass interface status changes through the routing socket (e.g. BSD 4.4). Note that GateD will also scan the interface list on receipt of a `SIGUSR2`.

interface `interface_list`

Sets interface options on the specified interfaces. An interface list is `all` or a list of interface names (see interface warning described above in “**options**”), domain names, or numeric addresses. Options available on this statement are:

precedence *`precedence`*

Sets the precedence for routes to this interface when it is up and appears to be functioning properly. The default precedence is `0`.

down precedence *`precedence`*

Sets the precedence for routes to this interface when GateD does not believe it to be functioning properly, but the kernel does not indicate it is down. The default value is `120`.

passive

Prevents GateD from changing the precedence of the route to this interface if it is not believed to be functioning properly due to lack of received routing information. GateD will only perform this check if the interface is actively participating in a routing protocol.

simplex

Defines an interface as unable to hear its own broadcast packets. Some systems define an interface as simplex with the IFF_SIMPLEX flag. On others it needs to be specified in the configuration file. On simplex interfaces, packets from myself are assumed to have been looped back in software, and are not used as an indication that the interface is functioning properly.

reject

Specifies that the address of the interface which matches these criteria will be used as the local address when installing **reject** routes in the kernel. Should only be used with systems based on BSD 4.3 Tahoe or earlier which have installed a **reject/blackhole** pseudo interface.

blackhole

Specifies that the address of the interface which matches these criteria will be used as the local address when installing **reject** routes in the kernel. Should only be used with systems based on BSD 4.3 Tahoe or earlier which have installed a **reject/blackhole** pseudo interface.

define *`address`*

Defines interfaces that might not be present when GateD is started so they may be referenced in the configuration file when `strictinterfaces` is defined.

Possible **define** keywords are:

broadcast *`address`*

Defines the interface as broadcast capable (e.g. Ethernet or Token Ring) and specifies the broadcast address.

pointtopoint *`address`*

Defines the interface as a point-to-point interface (e.g. SLIP or PPP) and specifies the address on the local side. The first *`address`* on the **define** statement references the address of the host on the **remote** end of the interface, the *`address`* specified after this **pointtopoint** keyword defines the address on the **local** side of the interface.

An interface not defined as **broadcast** or **pointtopoint** is assumed to be non-broadcast multi-access (NBMA), such as an X.25 network.

netmask `mask`

Specifies the subnet mask to be used on this interface. This is ignored on **pointtopoint** interfaces.

multicast

Specifies that the interface is multicast capable.

Interface lists

An interface list is a list of references to interfaces or groups of interfaces. There are four methods available for referring to interfaces. They are listed here from most general to most specific.

all

This refers to all available interfaces.

interface name wildcard

This refers to all the interfaces of the same type. Unix interfaces consist of the name of the device driver, like `ie`, and a unit number, like `0, 5` or `22`. References to the name contain only alphabetic characters and match any interfaces that have the same alphabetic part.

For example, `ie` on a Sun would refer to all Interlan Ethernet interfaces, `le` would refer to all Lance Ethernet interfaces. But `ie` would not match `iel0`.

Interface name

This refers to a specific interface, usually one physical interface. These are specified as an alphabetic part followed by a numeric part. This will match one specific interface. But be aware that on many systems, there can be more than one protocol (i.e. *IP*) address on a given physical interface. For example, `ef1` will match an interface named `ef1`, but not an interface named `ef10`.

Interface address

This matches one specific interface. The reference can be by protocol address (i.e. `10.0.0.51`), or by symbolic hostname (i.e. `nic.ddn.mil`). Note that a symbolic hostname reference is only valid when it resolves to only one address. Use of symbolic hostnames is not recommended.

If many interface lists are present in the configuration file with more than one parameter, these parameters are collected at run-time to create the specific parameter list for a given interface. If the same parameter is specified on more than one list, the parameter with the most specific interface is used.

For example, consider a system with three interfaces, `le0`, `le1` and `du0`.

```
rip yes {
    interface all noripin noripout ;
    interface le ripin ;
    interface le1 ripout ;
} ;
```

RIP packets would be accepted from interfaces `le0` and `le1`, but not from `du0`. RIP packets would only be sent on interface `le1`.

IP interface addresses and routes

The *BSD 4.3* and later networking implementations allow four types of interfaces. Some implementations allow multiple protocol addresses per physical interface, these are mostly based on *BSD 4.3 Reno* or later.

loopback

This interface must have the address of **127.0.0.1**.
Packets sent to this interface are sent back to the originator. This interface is also used as a catch-all interface for implementing other features, such as **reject** and **blackhole** routes. Although a netmask is reported on this interface, it is ignored. It is useful to assign an additional address to this interface that is the same as the OSPF or BGP router id. This allows routing to a system based on the router id which will work if some interfaces are down.

broadcast

This is a multi-access interface capable of a physical level broadcast such as Ethernet, Token Ring, or FDDI. This interface has an associated subnet mask and broadcast address. The interface route to a **broadcast** network will be a route to the complete subnet.

point-to-point

This is a tunnel to another host, usually on some sort of serial link. This interface has a local address, and a remote address. Although it may be possible to specify multiple addresses for a point-to-point interface, there does not seem to be a useful reason for doing so.

The remote address must be unique among all the interface addresses on a given router. The local address may be shared among many point-to-point and up to one non-point-to-point interface. This is technically a form of the router id method for address-less links. This technique conserves subnets as none are required when using this technique.

If a subnet mask is specified on a point-to-point interface, it is only used by RIP version 1 and HELLO to determine which subnets may be propagated to the router on the other side of this interface.

non-broadcast multi-access or **nbma**

This type of interface is multi-access, but not capable of broadcast. Examples would be frame relay and X.25. This type of interface has a local address and a subnet mask.

GateD ensures that there is a route available to each IP interface that is configured and up. Normally this is done by the **ifconfig** command that configures the interface; GateD does it to ensure consistency.

For point-to-point interfaces, GateD installs some special routes. If the local address on one or more point-to-point interfaces is not shared with a non-point-to-point interface, GateD installs a route to the local address pointing at the loopback interface with a preference of 110. This ensures that packets originating on this host destined for this local address are handled locally. OSPF prefers to route packets for the local interface across the point-to-point link where they will be returned by the router on the remote end. This is used to verify operation of the link. Since OSPF installs routes with a preference of 10, these routes will override the route installed with a preference of 110.

If the local address of one or more point-to-point interfaces is shared with a non-point-to-point interface, GateD installs a route to the local with a preference of 0 that will not be installed in the forwarding table. This is to prevent protocols like OSPF from routing packets to this address across a serial interface when this system could be functioning as a host.

When the status of an interface changes, GateD notifies all the protocols, which then take the appropriate action. GateD assumes that interfaces not marked UP do not exist. While this might not be the most correct action, it is the way things currently work.

GateD ignores any interfaces that have invalid data for the local, remote or broadcast addresses, or the subnet mask. Invalid data includes zeros in any field. GateD will also ignore any point-to-point interface that has the same local and remote addresses, it assumes it is in some sort of loopback test mode.

Definition statements

Definition statements are general configuration statements that relate to all of GateD or at least to more than one protocol. The five definition statements are **autonomoussystem**, **confederation**, **routerid**, **routing-domain**, and **martians**. When used, these statements must appear before any other type of configuration statement in the `gated.conf` file.

Also, one must use **confederation** and **routing-domain** together, but one cannot use **confederation**, **routing-domain**, and **autonomoussystem** together. They are mutally exclusive for configuring BGP.

Autonomous system configuration

autonomoussystem *`autonomous_system`* `[` **loops** *`number`* `] ;`

Sets the autonomous system number of this router to be *`autonomous_system`*. This option is required if BGP or EGP are in use. The AS number is assigned by the Network Information Center (NIC).

loops is only for protocols supporting AS paths, such as BGP. It controls the number of times this autonomous system may appear in an AS path and defaults to 1 (one).

Router ID configuration

routerid *`host`* `;`

Sets the router identifier for use by the BGP and OSPF protocols. The default is the address of the first interface encountered by GateD. The address of a non-point-to-point interface is preferred over the local address of a point-to-point interface. An address on a loopback interface that is not the loopback address (127.0.0.1) is most preferred.

Confederation

confederation *confederation* ;

Sets the confederation identifier for use by the BGP protocol for the configuration of group type confed. See the BGP statement for more information on configuring confederations (page 2-49). Note that **autonomoussystem** and **confederation** are mutually exclusive; they cannot both be defined within the same `gated.conf`. Also, note that **confederation** and **routing-domain** both must be set within the same `gated.conf`.

Routing domain identifier

routing-domain *rdi* ;

Sets the routing domain identifier for use by the BGP protocol for the configuration of group type confed. See the BGP statement for more information on configuring confederations (page 2-49). Note that **confederation** and **routing-domain** both must be set within the same `gated.conf`.

Martian configuration

```
martians {
    host host [ allow ] ;
    network [ allow ] ;
    network mask mask [ allow ] ;
    network ( masklen | / ) number [ allow ] ;
    default [ allow ] ;
} ;
```

Defines a list of **martian** addresses about which all routing information is ignored.

Sometimes a misconfigured system sends out obviously invalid destination addresses. These invalid addresses, called martians, are rejected by the routing software. This command allows additions to the list of martian addresses. See the section on Route Filtering for more information on specifying ranges (page 2-82).

Also, the **allow** parameter may be specified to explicitly allow a subset of a range that was disallowed.

Sample definition statements

```
options gendefault ;
autonomoussystem 249 ;
interface 128.66.12.2 passive ;
martians {
    0.0.0.26
};
```

The statements in the sample perform the following functions:

- The `options` statement tells the system to generate a default route when it peers with an EGP or BGP neighbor.
- The `autonomoussystem` statement tells GateD to use AS number 249 for in EGP and BGP.
- The `interface` statement tells GateD not to mark interface 128.66.12.2 as down even if it sees no traffic.
- The `martians` statement prevents routes to 0.0.0.26 from ever being accepted.

Protocol overview

Routing protocols determine the "best" route to each destination and distribute routing information among the systems on a network. Routing protocols are divided into two general groups: interior protocols and exterior protocols. GateD software combines management of the interior and exterior routing protocols in one software daemon.

Interior Routing Protocols

Interior protocols are used to exchange reachability information within an autonomous system (AS). They are referred to as a class by the acronym **igp**. There are two interior protocols currently supported by this version of GateD:

RIP

The Routing Information Protocol, Version 1 and Version 2, is the most commonly used interior protocol. RIP selects the route with the lowest metric as the best route. The metric is a hop count representing the number of gateways through which data must pass to reach its destination. The longest path that RIP accepts is 15 hops. If the metric is greater than 15, a destination is considered unreachable and GateD discards the route. RIP assumes the best route is the one that uses the fewest gateways i.e., the shortest path, not taking into account congestion or delay on route.

The RIP version 1 protocol is described in RFC 1058 and the RIP version 2 protocol is described in RFC 1388.

OSPF

Open Shortest Path First is a link-state protocol. OSPF is better suited than RIP for complex networks with many routers. OSPF provides equal cost multipath routing.

OSPF is described in RFC 1583, the MIB is defined in RFC 1253. Other related documents are RFC 1245, RFC 1246, and RFC 1370.

Exterior Routing Protocols

Exterior protocols are used to exchange routing information between autonomous systems. Exterior protocols are only required when an autonomous system must exchange routing information with another autonomous system. Routers within an autonomous system run an interior routing protocol like RIP. Only those gateways that connect an autonomous system to another autonomous system need to run an exterior routing protocol. There are two exterior protocols currently supported by GateD, EGP and BGP.

EGP

Exterior Gateway Protocol: Originally EGP reachability information was passed into ARPANET/MILNET "core" gateways where the best routes were chosen and passed back out to all connected autonomous systems. As the Internet moved toward a less hierarchical architecture, EGP, an exterior routing protocol which assumes a hierarchical structure, became less effective. The EGP protocol is described in RFC 827 and RFC 904.

BGP

Border Gateway Protocol is replacing EGP as the exterior protocol of choice. BGP exchanges reachability information between autonomous systems, but provides more capabilities than EGP. BGP uses path attributes to provide more information about each route as an aid in selecting the best route. Path attributes may include, for example, administrative preferences based on political, organizational, or security (policy) considerations in the routing decision. BGP supports non-hierarchical topologies and can be used to implement a network structure of equivalent autonomous systems.

BGP version 1 is described in RFC 1105, version 2 in RFC 1163, version 3 in RFC 1267, and version 4 in RFC 1771. The version 3 MIB is described in RFC 1269. The three documents, RFC 1164, RFC 1268, and RFC 1772, describe the application of versions 2, 3, and 4 in the Internet.

A protocol analysis of and experience with BGP version 3 are available in RFC 1265 and RFC 1266. RFC 1397 talks about advertising a default route in BGP version 2 and 3. And finally, RFC 1403 describes BGP - OSPF interaction.

Other Routing Protocols

Router Discovery

The Router Discovery protocol is used to inform hosts of the availability of hosts it can send packets to and is used to supplement a statically-configured default router. This is the preferred protocol for hosts to run, they are discouraged from *wiretapping* routing protocols. Router Discovery is described in RFC 1256.

Routing information protocol (RIP)

One of the most widely used interior gateway protocols is the Routing Information Protocol (RIP). RIP is an implementation of a distance-vector, or Bellman-Ford routing protocol for local networks. It classifies routers as active and passive (silent). Active routers advertise their routes (reachability information) to others; passive routers listen and update their routes based on advertisements, but do not advertise. Typically, routers run RIP in active mode, while hosts use passive mode.

A router running RIP in active mode broadcasts updates at set intervals. Each update contains paired values where each pair consists of an IP network address and an integer distance to that network. RIP uses a hop count metric to measure the distance to a destination. In the RIP metric, a router advertises directly connected networks at a metric of 1. Networks which are reachable through one other gateway are two hops, etc. Thus, the number of hops or hop count along a path from a given source to a given destination refers to the number of gateways that a datagram would encounter along that path. Using hop counts to calculate shortest paths does not always produce optimal results. For example, a path with hop count 3 that crosses three Ethernets may be substantially faster than a path with a hop count 2 that crosses two slow-speed serial lines. To compensate for differences in technology, many routers advertise artificially high hop counts for slow links.

As delivered with most UNIX systems, RIP is run by the routing daemon, **routed** (pronounced route-"d"). A RIP routing daemon dynamically builds on information received through RIP updates. When started up, it issues a REQUEST for routing information and then listens for responses to the request. If a system configured to supply RIP hears the request, it responds with a RESPONSE packet based on information in its routing database. The RESPONSE packet contains destination network addresses and the routing metric for each destination.

When a RIP RESPONSE packet is received, the routing daemon takes the information and rebuilds the routing database adding new routes and "better" (lower metric) routes to destinations already listed in the database. RIP also deletes routes from the database if the next router to that destination says the route contains more than 15 hops, or if the route is deleted. All routes through a gateway are deleted if no updates are received from that gateway for a specified time period. In general, routing updates are issued every 30 seconds. In many implementations, if a gateway is not heard from for 180 seconds, all routes from that gateway are deleted from the routing database. This 180-second interval also applies to deletion of specific routes.

RIP version 2 (more commonly known as RIP II) add additional capabilities to RIP. Some of these capabilities are compatible with RIP I and some are not. To avoid supplying information to RIP I routes that could be misinterpreted, RIP II can only use non-compatible features when its packets are multicast. On interfaces that are not capable of IP multicast, RIP I-compatible packets are used that do not contain potentially confusing information.

Some of the most notable RIP II enhancements are:

Next hop

RIP II can advertise a next hop other than the router supplying the routing update. This is quite useful when advertising a static route to a dumb router that does not run RIP. It avoids having packets destined through the dumb router from having to cross a network twice.

RIP I routers will ignore next hop information in RIP II packets. This may result in packets crossing a network twice, which is exactly what happens with RIP I. So this information is provided in RIP I-compatible RIP II packets.

Network mask

RIP I assumes that all subnetworks of a given network have the same network mask. It uses this assumption to calculate the network masks for all routes received. This assumption prevents subnets with different netmasks from being included in RIP packets. RIP II adds the ability to specify the network mask with each network in a packet.

While RIP I routers will ignore the network mask in RIP II packets, their calculation of the network mask will quite possibly be wrong. For this reason, RIP I-compatible RIP II packets must not contain networks that would be misinterpreted. These networks must only be provided in native RIP II packets that are multicast.

Authentication

RIP II packets may also contain one of two types of authentication strings that may be used to verify the validity of the supplied routing data. Authentication may be used in RIP I-compatible RIP II packets, but be aware that RIP I routers will ignore it.

The first method is a simple password in which an authentication key of up to 16 characters is included in the packet. If this does not match what is expected, the packet will be discarded. This method provides very little security as it is possible to learn the authentication key by watching RIP packets.

The second method is still experimental and may change in incompatible ways in future releases. This method uses the MD5 algorithm to create a crypto-checksum of a RIP packet and an authentication key of up to 16 characters. The transmitted packet does not contain the authentication key itself, instead it contains a crypto-checksum, called the *digest*. The receiving router will perform a calculation using the correct authentication key and discard the packet if the digest does not match. In addition, a sequence number is maintained to prevent the replay of older packets. This method provides a much stronger assurance that routing data originated from a router with a valid authentication key.

Two authentication methods can be specified per interface. Packets are always sent using the primary method, but received packets are checked with both the primary and secondary methods before being discarded. In addition, a separate authentication key is used for non-router queries.

RIP I and network masks

RIP I derives the network mask of received networks and hosts from the network mask of the interface via the packet which was received. If a received network or host is on the same natural network as the interface over which it was received and that network is subnetted (the specified mask is more specific than the *natural* netmask), the subnet mask is applied to the destination. If bits outside the mask are set, it is assumed to be a host, otherwise it is assumed to be a subnet.

On point-to-point interfaces, the netmask is applied to the remote address. The netmask on these interfaces is ignored if it matches the natural network of the remote address or is all ones.

Unlike in previous releases, the zero subnet mask (a network that matches the natural network of the interface, but has a more specific, or longer, network mask) is ignored. If this is not desirable, a route filter may be used to reject it.

The RIP statement

```
rip yes | no | on | off [ {
    broadcast ;
    nobroadcast ;
    nocheckzero ;
    preference preference ;
    defaultmetric metric ;
    query authentication [none | ([simple|md5] password)] ;
    interface interface_list
        [noripin] | [ripin]
        [noripout] | [ripout]
        [metricin metric]
        [metricout metric]
        [version 1]|[version 2 [multicast|broadcast]]
        [[secondary] authentication [none| ([simple|md5] password)]] ;
    trustedgateways gateway_list ;
    sourcegateways gateway_list ;
    traceoptions trace_options ;
} ] ;
```

The **rip** statement enables or disables RIP. If the **rip** statement is not specified, the default is "**rip off ;**". If enabled, RIP will assume **nobroadcast** when there is only one interface and `broadcast` when there is more than one.

The options are as follows:

broadcast

Specifies that RIP packets will be broadcast regardless of the number of interfaces present. This is useful when propagating static routes or routes learned from another protocol into RIP. In some cases, the use of **broadcast** when only one network interface is present can cause data packets to traverse a single network twice.

nobroadcast

Specifies that RIP packets will not be broadcast on attached interfaces, even if there are more than one. If a **sourcegateways** clause is present, routes will still be unicast directly to that gateway.

nocheckzero

Specifies that RIP should not make sure that reserved fields in incoming version 1 RIP packets are zero. Normally RIP will reject packets where the reserved fields are zero.

preference *preference*

Sets the preference for routes learned from RIP. The default preference is 0. This preference may be overridden by a preference specified in import policy.

defaultmetric *metric*

Defines the metric used when advertising routes via RIP that were learned from other protocols. If not specified, the default value is 16 (unreachable). This choice of values requires you to explicitly specify a metric in order to export routes from other protocols into RIP. This metric may be overridden by a metric specified in export policy.

query authentication [**none** | ([**simple**|**md5**] *password*)] ;

Specifies the authentication required of query packets that do not originate from routers. The default is **none**.

interface *interface_list*

Controls various attributes of sending RIP on specific interfaces. See the section on interface list specification for the description of the *interface_list* (page 2-14).

Note that if there are multiple interfaces configured on the same subnet, RIP updates will only be sent from the secondary interfaces if they are declared on the Interface statement by IP number (*not by logical interface name*). Also, note that IP aliases for the lo0 (Loopback) interface must also be declared by IP number. Under no circumstances will the export or designation of lo0 or 127.0.0.1 be allowed.

The possible parameters are:

noripin

Specifies that RIP packets received via the specified interface will be ignored. The default is to listen to RIP packets on all non-loopback interfaces.

ripin

This is the default. This argument may be necessary when **noripin** is used on a wildcard interface descriptor.

noripout

Specifies that no RIP packets will be sent on the specified interfaces. The default is to send RIP on all broadcast and non-broadcast interfaces when in **broadcast** mode. The sending of RIP on point-to-point interfaces must be manually configured.

ripout

This is the default. This argument is necessary when it is desired to send RIP on point-to-point interfaces and may be necessary when **noripin** is used on a wildcard interface descriptor.

metricin *metric*

Specifies the RIP metric to add to incoming routes before they are installed in the routing table. The default is the kernel interface metric plus 1 (which is the default RIP hop count). If this value is specified, it will be used as the absolute value, the kernel metric will not be added. This option is used to make this router prefer RIP routes learned via the specified interface(s) less than RIP routes from other interfaces.

metricout *metric*

Specifies the RIP metric to be added to routes that are sent via the specified interface(s). The default is zero. This option is used to make other routers prefer other sources of RIP routes over this router.

version 1

Specifies that the RIP packets sent on the specified interface(s) will be version 1 packets. This is the default.

version 2

Specifies that RIP version 2 packets will be sent on the specified interfaces(s). If IP multicast support is available on the specified interface(s), the default is to send full version 2 packets. If multicast support is not available, version 1 compatible version 2 packets will be sent.

multicast

Specifies that RIP version 2 packets should be multicast on this interface. This is the default.

broadcast

Specifies that RIP 1-compatible RIP version 2 packets should be broadcast on this interface, even if IP multicast is available.

[**secondary**] **authentication** [**none** | ([**simple**|**md5**] *password*)]

This defines the authentication type to use. It applies only to RIP version 2 and is ignored for RIP 1 packets. The default authentication type is **none**. If a password is specified, the authentication type defaults to **simple**. The password should be a quoted string with between 0 and 16 characters.

If **secondary** is specified, this defines the secondary authentication. If omitted, the primary authentication is specified. The default is primary authentication of **none** and no secondary authentication.

trustedgateways *gateway_list*

Defines the list of gateways from which RIP will accept updates. The *gateway_list* is simply a list of host names or IP addresses. By default, all routers on the shared network are trusted to supply routing information. But, if the **trustedgateways** clause is specified, only updates from the gateways in the list are accepted.

sourcegateways *gateway_list*

Defines a list of routers to which RIP sends packets directly, not through multicast or broadcast. This can be used to send different routing information to specific gateways. Updates to gateways in this list are not affected by **noripout** on the interface.

traceoptions *trace_options*

Specifies the tracing options for RIP. (See Trace Statements (page 2-9) and the RIP-specific tracing options below.)

Tracing options

The **policy** option logs information whenever a new route is announced, or the metric being announced changes, or a route goes or leaves holddown.

Packet tracing options (which may be modified with **detail**, **send** or **recv**):

packets

All RIP packets.

request

RIP information request packets, such as **REQUEST**, **POLL** and **POLLENTRY**

response

RIP **RESPONSE** packets, which are the type of packet that actually contains routing information.

other

Any other type of packet. The only valid ones are **TRACE_ON** and **TRACE_OFF**, both of which are ignored.

The OSPF protocol

Open Shortest Path Routing (OSPF) is a *shortest path first* (SPF) or *link-state* protocol. OSPF is an interior gateway protocol that distributes routing information between routers in a single autonomous system (AS). OSPF chooses the least cost path as the best path. Suitable for complex networks with a large number of routers, OSPF provides equal cost multipath routing where packets to a single destination can be sent via more than one interface simultaneously. In a link-state protocol, each router maintains a database describing the entire AS topology, which it builds out of the collected link state advertisements of all routers. Each participating router distributes its local state (i.e., the router's usable interfaces and reachable neighbors) throughout the AS by flooding. Each multi-access network that has at least two attached routers has a *designated router* and a *backup designated router.* The designated router floods a link state advertisement for the multi-access network and has other special responsibilities. The designated router concept reduces the number of adjacencies required on a multi-access network.

OSPF allows networks to be grouped into areas. Routing information passed between areas is abstracted, potentially allowing a significant reduction in routing traffic. OSPF uses four different types of routes, listed in order of preference:

- intra-area
- inter-area
- type 1 external
- type 2 external

Intra-area paths have destinations within the same area, inter-area paths have destinations in other OSPF areas and Autonomous System External (ASE) routes are routes to destinations external to the AS. Routes imported into OSPF as type 1 routes are supposed to be from IGPs whose external metrics are directly comparable to OSPF metrics. When a routing decision is being made, OSPF will add the internal cost to the AS Border router to the external metric. Type 2 ASEs are used for EGPs whose metrics are not comparable to OSPF metrics. In this case, only the internal OSPF cost to the AS Border router is used in the routing decision.

From the topology database, each router constructs a tree of the shortest paths with itself as the root. This shortest-path tree gives the route to each destination in the AS. Externally derived routing information appears on the tree as leaves. The link-state advertisement format distinguishes between information acquired from external sources and information acquired from internal routers, so there is no ambiguity about the source or reliability of routes. Externally derived routing information (for example, routes learned from EGP or BGP) is passed transparently through the autonomous system and is kept separate from OSPF's internally derived data. Each external route can also be tagged by the advertising router, enabling a passing of additional information between routers on the borders of the autonomous system.

OSPF optionally includes *type of service* (TOS) routing and allows administrators to install multiple routes to a given destination for each type of service (e.g. low delay or high throughput.) A router running OSPF uses the destination address and the type of service to choose the best route to the destination.

OSPF intra- and inter-area routes are always imported into the GateD routing database with a preference of 10. It would be a violation of the protocol if an OSPF router did not participate fully in the area's OSPF, so it is not possible to override this. Although it is possible to give other routes lower preference values explicitly, it is ill-advised to do so.

Hardware multicast capabilities are also used where possible to deliver link-status messages. OSPF areas are connected by the *backbone* area, the area with identifier 0.0.0.0. All areas must be logically contiguous and the backbone is no exception. To permit maximum flexibility, OSPF allows the configuration of *virtual* links enable the backbone area to appear contiguous despite the physical reality.

All routers in an area must agree on that area's parameters. A separate copy of the link-state algorithm is run for each area. Because of this, most configuration parameters are defined on a per area basis. All routers belonging to an area must agree on that area's configuration. Mis-configuration will lead to adjacencies not forming between neighbors, and routing information might loop or not flow.

Authentication

All OSPF protocol exchanges are authenticated. Authentication guarantees that routing information is only imported from trusted routers, to protect the Internet and its users. A variety of authentication schemes can be used, but a single scheme must be configured for each area. This enables some areas to use much stricter authentication than others. OSPF protocol exchanges may be authenticated. Authentication guarantees that routing information is imported only from trusted routers, to protect the Internet and its users. There are two authentication schemes available. The first uses a simple authentication key of up to 8 characters and is standardized. The second is still experimental and uses the MD5 algorithm and an authentication key of up to 16 characters.

The simple password provides very little protection because in many cases it is possible to easily capture packets from the network and learn the authentication key. The experimental MD5 algorithm provides much more protection as it does not include the authentication key in the packet.

The OSPF specification currently specifies that the authentication type be configured per area with the ability to configure separate passwords per interface. This has been extended to allow the configuration of different authentication types and keys per interface. In addition, it is possible to specify both a *primary* and a *secondary* authentication type and key on each interface. Outgoing packets use the primary authentication type, but incoming packets may match either the primary or secondary authentication type and key.

The OSPF statement

```
ospf yes | no | on | off [ {
    defaults {
        precedence precedence ;
        cost cost ;
        tag [ as ] tag ;
        type 1 | 2 ;
        inherit-metric ;
    } ;
    exportlimit routes ;
    exportinterval time ;
    traceoptions trace_options ;
    monitorauthkey authkey ;
    monitorauth none | ( [ simple | md5 ] authkey ) ;
    backbone | ( area area ) {
        authtype  none | simple ;
        stub [ cost cost] ;
        networks {
            network [ restrict ] ;
            network mask mask [ restrict ] ;
            network ( masklen | / ) number [ restrict ] ;
            host host [ restrict ] ;
        } ;
        stubhosts {
            host  cost cost ;
        } ;
        interface interface_list; [cost cost ] {
            interface_parameters
        } ;
        interface interface_list nonbroadcast [cost cost ] {
            pollinterval time ;
            routers {
                gateway [ eligible ] ;
            } ;
            interface_parameters
        } ;
        Backbone only:
        virtuallink neighborid router_id transitarea area {
            interface_parameters
        } ;
    } ;
} ] ;
```

The following are the *interface_parameters* referred to above. These may be specified on any class of interface and are described under the **interface** clause.

enable | **disable** | **passive** ;
retransmitinterval *time* ;
transitdelay *time* ;
priority *priority* ;
hellointerval *time* ;
routerdeadinterval *time* ;
authkey *auth_key* ;

defaults

These parameters specify the defaults used when importing OSPF ASE routes into the GateD routing table and exporting routes from the GateD routing table into OSPF ASEs.

protocol-precedence *precedence*

Sets the global precedence for BGP incoming routes.. This precedence may be overridden by a precedence specified on the group or peer statement or by import policy.   The default precedence is 170

cost *cost*

The cost is used when exporting a non-OSPF route from the GateD routing table into OSPF as an ASE. This may be explicitly overridden in export policy by the **metric** keyword. The default value is 1.

Note: This parameter has nothing to do with the “**cost**” keyword on the interface command.

tag [**as**] *tag*

OSPF ASE routes have a 32-bit tag field that is not used by the OSPF protocol, but may be used by export policy to filter routes. When OSPF is interacting with an EGP, the tag field may be used to propagate AS path information, in which case the **as** keyword is specified and the tag is limited to 12 bits of information. If not specified, the tag is set to zero.

type *1* | *2*

Routes exported from the GateD routing table into OSPF default to becoming type 2 ASEs. This default may be explicitly changed here and overridden in export policy.

inherit-metric

Inherit-metric allows an OSPF ASE route to inherit the metric of the external route when no metric is specified on the export. This option maintains compatibility with all the current export functions:

- A metric specified on the export will take precedence.
- The cost specified in the default will be used if inherit-metric is not specified.

ASE export rate

Because of the nature of OSPF, the rate at which ASEs are flooded must be limited. These two parameters can be used to adjust those rate limits.

exportinterval *time*

This specifies how often a batch of ASE link state advertisements will be generated and flooded into OSPF. The default is once per second.

exportlimit *routes*

This parameter specifies how many ASEs will be generated and flooded in each batch. The default is 100.

traceoptions *trace_options*

Specifies the tracing options for OSPF. (See Trace Statements (page 2-9) and the OSPF specific tracing options below.)

monitorauthkey *authkey*

OSPF state may be queried using the **ospf_monitor** (This should be a hyperlink) utility. This utility sends non-standard OSPF packets which generate a text response from OSPF. By default, these requests are not authenticated. If an authentication key is configured, the incoming requests must match the specified authentication key. No OSPF state may be changed by these packets, but the act of querying OSPF can utilize system resources.

backbone
area *area*

Each OSPF router must be configured into at least one OSPF area. If more than one area is configured, at least one must be the backbone. The backbone may only be configured using the **backbone** keyword, it may not be specified as *area* 0. The backbone interface may be a **virtuallink**.

authtype none | **simple**

OSPF specifies an authentication scheme per area. Each interface in the area must use this same authentication scheme although it may use a different **authenticationkey**. The currently valid values are **none** for no authentication, or **simple** for simple password authentication. The default is **none**.

If **authkey simple** is specified but there is no **authkey** *xxx* in the `/etc/gated.conf` file, this is equivalent to specifying **authkey none**. In both cases, authentication is disabled.

stub [**cost** *cost*]

A **stub** area is one in which there are no ASE routes. Specifying **stub** will prevent participating routers from advertising ASE routes into that area. If a **cost** is specified, this is used to inject a default route into the area with the specified *cost*. This is not valid for backbone area. The **stub** keyword must be specified on all routers within a given area.

networks

The **networks** list describes the scope of an area. Intra-area LSAs that fall within the specified ranges are not advertised into other areas as inter-area routes. Instead, the specified ranges are advertised as *summary network* LSAs. If **restrict** is specified, the summary network LSAs are not advertised. Intra-area LSAs that do not fall into any range are also advertised as summary network LSAs. This option is very useful on well-designed networks in reducing the amount of routing information propagated between areas. The entries in this list are either networks, or a subnetwork/mask pair. See the section on Route Filtering for more detail about specifying ranges (page 2-82).

stubhosts

This list specifies directly-attached hosts that should be advertised as reachable from this router and the costs they should be advertised with. Point-to-point interfaces on which it is not desirable to run OSPF should be specified here.

It is also useful to assign a additional address to the loopback interface (one not on the 127 network) and advertise it as a stubhosts. If this address is the same one used as the `router-id`, it enables routing to OSPF routers by `router-id`, instead of by interface address. This is more reliable than routing to one of the router's interface addresses which may not always be reachable.

interface *interface_list* [**cost** *cost*]

This form of the interface clause is used to configure a **broadcast** (which requires IP multicast support) or a **point-to-point** interface. See the section on Interface list specification (page 2-14) for the description of the *interface_list*.

Each interface has a ***cost***. The costs of all interfaces a packet must cross to reach a destination are summed to get the cost to that destination. The default cost is one, but another non-zero value may be specified.

Interface parameters common to all types of interfaces are:

enable | **disable** | **passive**

Enable or disable OSPF on the interface. If **passive** is used, OSPF behaves as though it is enabled on the interface (the interface's routes are included in LSAs issued by the router), but OSPF packets are not sent or accepted on the interface. This may be useful for advertising DMZs or other interconnect networks into OSPF, as an alternative to advertising them into ospfase.

retransmitinterval *time*

The number of seconds between link state advertisement retransmissions for adjacencies belonging to this interface.

transitdelay *time*

The estimated number of seconds required to transmit a link state update over this interface. Transitdelay takes into account transmission and propagation delays and must be greater than 0.

priority *priority*

A number between 0 and 255 specifying the priority for becoming the designated router on this interface. When two routers attached to a network both attempt to become the designated router, the one with the highest priority wins. A router whose router priority is set to 0 is ineligible to become designated router.

OSPF supports both NBMA and P2P interfaces. The priority for these interfaces must be manually configured to elect the designated router.

If no priority keyword is present, priority defaults to 0. At least one router on a network must have a priority if OSPF is to work at all.

hellointerval *time*

The length of time, in seconds, between Hello packets that the router sends on the interface.

routerdeadinterval *time*

The number of seconds not hearing a router's Hello packets before the router's neighbors will declare it down.

authkey *auth_key*

Used by OSPF authentication to generate and verify the authentication field in the OSPF header. The authentication key can be configured on a per-interface basis. It is specified by one to eight decimal digits separated by periods, or a one- to eight-character string in double quotes. The password “01.02.03” is equivalent to “1.2.3” since the fields are treated as numeric values, not just as an ASCII string. Similarly, the password “0” is equivalent to “0.0.0.0”.

If no **authkey** keyword is present, authentication will be disabled exactly as if **authtype none** has been specified.

Point-to-point interfaces also support this additional parameter:

nomulticast

By default, OSPF packets to neighbors on point-to-point interfaces are sent via the IP multicast mechanism. Although some implementations of IP multicasting for Unix have a bug that precludes the use of IP multicasting on these interfaces. GateD will detect this condition and fall back to sending unicast OSPF packets to this point-to-point neighbor.

If the use of IP multicasting is not desired because the remote neighbor does not support it, the **nomulticast** parameter may be specified to force the use of unicast OSPF packets. This option may also be used to eliminate warnings when GateD detects the bug mentioned above.

interface *interface_list* **nonbroadcast** [**cost** *cost*]

This form of the interface clause is used to specify a **nonbroadcast** interface on a **non-broadcast multi-access** (NBMA) medium. Since an OSPF **broadcast** medium must support IP multicasting, a broadcast-capable medium, such as Ethernet, that does not support IP multicasting must be configured as a non-broadcast interface.

A non-broadcast interface supports any of the standard **interface** clauses listed above, plus the following two that are specific to non-broadcast interfaces:

pollinterval *time*

Before adjacency is established with a neighbor, OSPF packets are sent periodically at the specified **pollinterval**.

routers

By definition, it is not possible to send broadcast packets to discover OSPF neighbors on a non-broadcast medium, so all neighbors must be configured. The list includes one or more neighbors and an indication of their eligibility to become a designated router.

virtuallink neighborid *router_id* **transitarea** *area*

Virtual links are used to establish or increase connectivity of the backbone area. The **neighborid** is the *router_id* of the other end of the virtual link. The transit *area* specified must also be configured on this system. All standard interface parameters defined by the **interface** clause above may be specified on a virtual link.

Tracing options

In addition to the following OSPF specific trace flags, OSPF supports the **state** which traces interface and neighbor state machine transitions.

lsabuild

Link State Advertisement creation

spf

Shortest Path First (SPF) calculations

Packet tracing options (which may be modified with **detail**, **send** and **recv**):

hello

OSPF **HELLO** packets which are used to determine neighbor reachability.

dd
OSPF Database Description packets which are used in synchronizing OSPF databases.

request
OSPF Link State Request packets which are used in synchronizing OSPF databases.

lsu
OSPF Link State Update packets which are used in synchronizing OSPF databases.

ack
OSPF Link State Ack packets which are used in synchronizing OSPF databases.

IS-IS Intra-Domain Protocol

IS-IS is a link state interior gateway protocol (IGP) originally developed for routing ISO/CLNP (International Organization for Standardization/Connectionless Network Protocol) packets. The version distributed with GateD can route IP packets as well.

In ISO terminology, a router is referred to as an "intermediate system" (IS). IS-IS intra-domain routing is organized hierarchically so that a large domain may be administratively divided into smaller areas using level 1 intermediate systems within areas and level 2 intermediate systems between areas.

Routing between administrative domains is handled by Border Intermediate Systems (BISs) using IDRP, the inter-domain routing protocol. Level 1 systems route directly to systems within their own area and route toward a Level 2 Intermediate System when the destination system is in a different area. Level 2 Intermediate Systems route between areas and keep track of the paths to destination areas. Systems in the Level 2 subdomain route towards a destination area, or another routing domain. As with any internet routing protocol, IS-IS support for large routing domains may also include many types of individual subnetworks. These subnetworks may include point-to-point links, multipoint links and broadcast subnetworks like ISO 8802 LANs.

In IS-IS, all subnetwork types are treated by the subnetwork independent functions as though they were connectionless subnetworks using subnetwork convergence functions where necessary. Like OSPF, IS-IS uses a "shortest-path first" algorithm to determine routes. GateD configuration syntax allows as much autoconfiguration as possible, reducing the probability of error.

This integration also allows the ability to specify policy for exchanging routing information with other protocols running in GateD.

IS-IS Statement

This statement enables the IS-IS protocol in GateD and configures the interfaces that are to run IS-IS. By default, IS-IS is disabled. The IS-IS statement consists of an initial description of the Intermediate System and a list of statements that determine the configuration of the specific circuits and networks to be managed.

```
isis no | ip {
   level 1|2 ;
   [traceoptions <isis_traceoptions> ;]
   [systemid <string> ;]
   [area <string> ;]
   [set <isis_parm> value ;]
   circuit|interface <interface-name>
       metric [level 1|2] metric
       priority [level 1|2] priority pointopoint ;
   [ipreachability level (1|2) (internal|external|summary)
       ipaddr netmask [metric metric;]]
} ;
```

Statements may appear in any order and include:

level

Indicates whether GateD is running on a Level 1 (intra-area) or Level 2 (inter-area) IS. The default is Level 1.

traceoptions

These are covered in the Trace Options section below.

systemid *string*

This is a mandatory parameter. If no system identifier is specified, GateD will report an error "`systemID is not set`", and abort. User can specify **systemid** in two ways, as a hex string or as a character string. If the string starts with 0x or 0X, GateD interprets it as hex string. If it does not contain 0x or 0X, it interprets it as a character string. However, the trace file prints all the values as hex strings.

area *string*

This is a mandatory parameter. IS-IS area addresses are configured based on this parameter. If **area** is not specified in the configuration file, GateD will report an error "area address is not set", and aborts. User can specify **area** in two ways - as a hex string or as a character string. If the string starts with 0x or 0X, GateD interprets it as hex string. If it does not contain 0x or 0X, it interprets as character string. However, the trace file prints all the values as hex strings.

circuit | interface *interface-name*

Each **circuit** statement specifies one of the circuits or interfaces the system will manage. Circuits normally correspond to UNIX interfaces, with *string* being the interface name. The **circuit** attributes are a list of options that may appear in any order in the **circuit** statement.

metric level [**1** | **2**] *metric*

Allows specifications of Level 1 and Level 2 metrics for each circuit. Only the default metric type is supported. IS-IS metrics must be in the range 1 to 63. If no metric is set for the circuit, the default value is 63. If only level 2 is specified, then this circuit is configured to run level 2 only, not level 1 at all.

priority [**level** [**1** | **2**] *priority*]

Determines designated router election results; higher values give a higher likelihood of becoming the designated router. The level defaults to Level 1. If no priority is specified, priority is set to a random value between 0 and 127.

On a level 2 IS, to configure a circuit with a Level 1 metric of 10 and a Level 2 metric of 20, add two metric options to the circuit statement.

The default Level is 1; the default metric is 63. The default precedence for IS-IS Level 1 is 15. For IS-IS Level 2, the default precedence is 18.

pointopoint

Allows this circuit to be point-to-point type. User has to specify this if they want this circuit to be a point-to-point type for IS-IS.

ipreachability level (**1** | **2**) **internal** | **external** | **summary** *ipaddr netmask* [**metric** metric]

The **ipreachability** statement is used to specify IP networks that will be advertised as reachableby the IS-IS System. There are five parts to the options - the keyword ipreachability, the level at which the route has to be advertised, the type of reachability (internal, external or summary), the network (specified by IP address and network mask), and the metric associated with the network.

All, with the exception of metric, are mandatory. IP addresses and masks are specified in dot notation. These networks are assumed to be directly attached networks to this router. If this is not a directly-attached network, the network will not advertised in the IS-IS packets. **ipreachability** with the **summary** keyword is used to summarize the networks.

set isis_parm *integer*

The following list shows the names and default values of the variables that can be changed using the **set** statement. These may appear in any order in the **isis** statement.

origL1LSPBufSize *integer*

Allows you to set originating Level 1 LSP Buffer size to given number of bytes. The default value is 1497 octets. This size must be less than or smallest MTU of any link the network. Normally 1497 is sufficient unless the MTU of a link is lowered to be less than 1497. All the routers in the network has to be specified the same size of LSP. Otherwise, the results are not predictable.

origL2LSPBufSize *integer*

Allows you to set originating Level 2 LSP Buffer size to given number of bytes. The default value is 1497 octets. This size must be less than or smallest MTU of any link the network. Normally 1497 is sufficient unless the MTU of a link is lowered to be less than 1497. All the routers in the network has to be specified the same size of LSP. Otherwise, the results are not predictable.

dataLinkBlocksize *integer*

Allows you to set the maximum size of Hello packets to given number of bytes. The default value is 1497 octets. This size must be less than or smallest MTU of any link in the network. Normally 1497 is sufficient unless the MTU of a link is lowered to be less than 1497. All the routers in the network has to be specified the same size of LSP. Otherwise, the results are not predictable.

sysHoldingTimer *integer*

Specifies the multiplier for the hold timer in Hello packets. The value specified by **sysHoldingTimer** multiplied by the **sysIIHInterval** gives the holding time for Hello packets. The default value for **sysHoldingTimer** is 3. If the receiving system does not receive a hello packet by the expiration of the holding time, it will drop the adjacency formed with the transmitting system.

sysISHInterval *integer*

Specifies the transmit frequency of ISH PDUs on a point-to-point circuit. The default value is 10 seconds.

sysIIHInterval *integer*

Specifies the transmit frequency of IIH PDUs on a broadcast circuit. The default value is 3 seconds.

minLSPGenInterval *integer*

Specifies the minimum interval between sucessive LSP generations. The default value is 30 seconds.

maxLSPGenInterval *integer*

Specifies the maximum interval between successive LSP generations. The default value is 900 seconds.

minLSPXmitInterval *integer*

Specifies the minimum LSP transmission interval on a point to point circuit. The default value is 2 seconds.

minBLSPXmitinterval *integer*

Specifies the minimum LSP transmission interval on a broadcast circuit. The default value is 5 seconds.

BLSPThrottle *integer*

Specifies the maximum number of LSPs sent per **minBLSPXmitInterval**. The default value is 50 seconds.

maximumAge *integer*

Specifies the initial (and maximum) life time of any LSP. This is stored in the lifetime field in the LSP. The default value is 1200 seconds.

completeSNPInterval *integer*

Specifies the transmit frequency of CSNPs. The default value is 10 seconds.

partialSNPInterval *integer*

Specifies the transmit frequency of PSNPs. The default value is 2 seconds.

zeroAgeLifetime *integer*

Specifies the time a LSP with zero lifetime is held before purging it from the LSP database. The default value is 60 seconds.

dumpDBinterval *integer*

Specifies the frequency with which LSP database is traced. The default value is 30 seconds.

Tracing options

IS-IS has its own internal tracing flags which are distinct from the flags maintained globally by GateD. The mechanism to set tracing is via the **isis traceoptions** statement, in the form:

```
traceoptions ["trace_file" [replace][size size [k|m] files files]]
isis_trace_options;
```

isis_trace_options can include one or more of the following:

all	-	everything below
iih	-	IIHs sent and received
lanadj	-	lan adjacency updates
p2padj	-	point to point adjacency updates
lspdb	-	signatures in the LSP database
lspcontent	-	contents of LSPs in the database
lspinput	-	input processing of LSPs
flooding	-	flooding of all LSPs
buildlsp	-	generation of local LSPs
csnp	-	processing and construction of CSNPs
psnp	-	processing and construction of PSNPs
route	-	route changes
update	-	individual routes changed
paths	-	route paths as calculated by spf algorithm
spf	-	running of spf algorithm
events	-	interesting protocol events

IS-IS configuration task lists

To configure IS-IS, the user must complete the following tasks. These tasks are mandatory. Other parameters are optional, and GateD will use default values. Users might want to set the other parameters depending on their network topology.

– configure ISO address for the interfaces (in `grifconfig.conf`)

– configure PPP, ATM and Frame Relay for IS-IS

– enable IS-IS in `gated.conf`

– configure **systemid** in `gated.conf`

– configure **area** in `gated.conf`

– configure **interface** in `gated.conf`

Configure ISO address for the interface

This is a mandatory task.

Assign ISO address for the interface. You can come up with ISO address by appending systemid to the **area**. For example, if the **area** address is 49.0000.80 and **systemid** you assigned for the interface is 3260.3260.3260, ISO address becomes 49.0000.80.3260.3260.3260.00. You have to include the last 00 byte when you are configuring ISO address. This is NSEL parameter specified in the ISO addresses. However, the NSEL parameter is not part of the **systemid** in `gated.conf`. Each system should have a unique **systemid**. If you configure two systems with the same**systemid**, the results will be unpredictable. You can add this address in `grifconfig.conf` file.

The syntax for adding the address in `grifconfig.conf` is

```
<interface-name>  <iso-address> <iso-area> - iso
```

Example:

```
gf030 49.0000.80.3260.3260.3260.00 49.0000.80 - iso
```

Configure PPP, ATM and Frame Relay for IS-IS

If you are using PPP, Frame Relay or ATM interfaces, you need to follow this task.

This is a mandatory task. For PPP, you have to add "`enable osinlcp`" in `/etc/grppp.conf` file in order to make IS-IS work over PPP. This is just like existing "`enable ipcp`" line in `/etc/grppp.conf`.

For Frame Relay mode, you have to edit `/etc/grfr.conf` file and edit the PVC line to add `ISIS=Y`. This enables the PVC to start accepting IS-IS packets.

For ATM interfaces, you have to edit `/etc/gratm.conf` file and add `proto=isis` or `proto=isis_ip` for the desired PVC to accept IS-IS packets. The following examples show the additional configurations necessary to enable IS-IS over the media.

Example for grppp.conf:

```
enable osinlcp
```

Example for grfr.conf:

```
pvc gs030 101 192.0.2.99 Enabled=Y Name="Router2" ISIS=Y
```

Example for gratm.conf:

```
pvc ga030 0/40 proto=isis traffic_shape=high_speed_high_quality
pvc ga030 0/41 proto=isis_ip traffic_shape=high_speed_high_quality
```

Enable IS-IS in gated.conf:

This is a mandatory task. IS-IS is enabled by using "**isis yes**" or "**isis ip**" in GateD configuration file.

Example:

```
isis yes {
} ;
```

This statement enables the IS-IS protocol in GateD. By default IS-IS is disabled. The above statement alone will not bring up IS-IS. GateD will abort with "`systemid is not set`" message. **systemid** and **area** are mandatory parameters for IS-IS to run.

Configure area in gated.conf

This is a mandatory task. **area** can be specified as a hex string or as a character string in GateD configuration. Two routers in Level-1 mode communicate with each other only if they belong to the same area. Two level-2 routers can communicate across area boundaries.

Example 1:

```
isis yes {
       area "0x49000080";
} ;
```

Example 2:

```
isis yes {
        area "aaaa";
        # The above string will be printed in the trace file as 61.6161.61
} ;
```

Configure systemid in gated.conf

This is a mandatory task. **systemid** is specified as a hex string or as a character string in GateD configuration. Each router in the network should have a unique **systemid** and should be 6 characters long. **systemid** has to be specified as 12 hex digits in the hex string format or as 6 characters in the string format. As with **area**, the character string will be printed in the hex format in trace file.

Example:

```
isis yes {
        area "0x49000080";
        systemid "0x326032603260";
} ;
```

Example:

```
isis yes {
        area "0x49000080";
        systemid "aaaaaa";
} ;
```

Configure interface in gated.conf

This is a mandatory task. If user does not configure **interface** in GateD configuration file, the interface will not be transmitting/receiving IS-IS packets. **interface** can be enabled by using either `interface` or `circuit` keyword. User can specify any of the optional `circuit` parameters allowed on this statement.

Example:

```
isis yes {
       area "49000080";
       systemid "326032603260";
       interface "gf030" metric 10 priority 60;
} ;
```

The Exterior Gateway Protocol (EGP)

The Exterior Gateway Protocol (EGP) is an exterior routing protocol used for exchanging routing information with gateways in other autonomous systems. Unlike interior protocols, EGP propagates only reachability indications, not true metrics. EGP updates contain metrics, called *distances* which range from 0 to 255. GateD will only compare EGP distances learned from the same AS.

Before EGP sends routing information to a remote router, it must establish an adjacency with that router. This is accomplished by an exchange of *Hello* (not to be confused with the HELLO protocol, or OSPF HELLO messages) and *I Heard You* (I-H-U) messages with that router.

Computers communicating via EGP are called EGP *neighbors*, and the exchange of HELLO and I-H-U messages is referred to as *acquiring a neighbor.* Once the neighbor is acquired, the system *polls* the neighbor for routing information. The neighbor responds by sending an *update* containing routing information. If the system receives a poll from its neighbor, it responds with its own update packet. When the system receives an update, it includes routes from the update into its routing database. If the neighbor fails to respond to three consecutive polls, GateD assumes that the neighbor is down and removes the neighbor's routes from its database.

The EGP statement

```
egp yes | no | on | off
[ {
    preference preference ;
    defaultmetric metric ;
    packetsize number ;
    traceoptions trace_options ;
    group
        [ peeras autonomous_system ]
        [ localas autonomous_system ]
        [ maxup number ]
    {
        neighbor host
            [ metricout metric ]
            [ preference preference ]
            [ preference2 preference ]
            [ ttl ttl ]
            [ nogendefault ]
            [ importdefault ]
            [ exportdefault ]
            [ gateway gateway ]
            [ lcladdr local_address ]
            [ sourcenet network ]
            [ minhello | p1 time ]
            [ minpoll | p2 time ]
            [ traceoptions trace_options ]
            ;
   } ;
} ] ;
```

preference *preference*

Sets the preference for routes learned from RIP. The default preference is 200. This preference may be overridden by a preference specified on the `group` or `neighbor` statements, or by import policy.

defaultmetric *metric* ;

Defines the metric used when advertising routes via EGP. If not specified, the default value is 255 which some systems may consider unreachable. This choice of values requires you to specify a metric when exporting routes to EGP neighbors. This metric may be overridden by a metric specified on the neighbor or group statements, or in export policy.

packetsize *maxpacketsize*

This defines the expected maximum size of a packet that EGP expects to receive from this neighbor. If a packet larger than this value is received, it will be incomplete and have to be discarded. The length of this packet will be noted and the expected size will be increased to be able to receive a packet of this size. Specifying the parameter here will prevent the first packet from being dropped. If not specified, the default size is 8192 bytes. All packet sizes are rounded up to a multiple of the system page size.

traceoptions *trace_options*

Specifies the tracing options for EGP. By default these are inherited from the global trace options. These values may be overridden on a group or neighbor basis. (See the Trace Statements (page 2-9) and the EGP specific tracing options below.)

group

EGP neighbors must be specified as members of a **group**. A group is usually used to group all neighbors in one autonomous system. Parameters specified on the group clause apply to all of the subsidiary neighbors unless explicitly overridden on a **neighbor** clause. Any number of group clauses may specify any number of neighbor clauses.

Any parameters from the group subclause may be specified on the group clause to provide defaults for the whole group (which may be overridden for individual neighbors). In addition, the group clause is the only place to set the following attributes:

peeras

Identifies the autonomous system number expected from peers in the group. If not specified, it will be learned dynamically.

localas

Identifies the autonomous system which GateD is representing to the group. The default is that which has been set globally in the **autonomoussystem** statement. This option is usually only used when *masquerading* as another autonomous system, and its use is discouraged.

maxup

Specifies the number of neighbors GateD should acquire from this group. The default is to acquire all of the neighbors in the group. GateD will attempt to acquire the first **maxup** neighbors in the order listed. If one of the first neighbors is not available, it will acquire one further down the list. If after start-up GateD does manage to acquire the more desirable neighbor, it will drop the less desirable one.

neighbor *neighbor_address*

Each neighbor subclause defines one EGP neighbor within a group. The only part of the subclause that is required is the `neighbor_address` argument which is the symbolic host name or IP address of the neighbor. All other parameters are optional.

preference *preference*

Specifies the preference used for routes learned from these neighbors. This can differ from the default EGP preference set in the **egp** statement, so that GateD can prefer routes from one neighbor, or group of neighbors, over another. This preference may be explicitly overridden by import policy.

preference2 *preference*

In the case of a preference tie, the second preference, **preference2,** may be used to break the tie. The default value is 0.

metricout *metric*

This defines a metric to be used for all routes sent to this neighbor. The value overrides the default metric set in the **egp** statement and any metrics specified by export policy, but only for this specific neighbor or group of neighbors.

nogendefault

Prevents GateD from generating a default route when EGP receives a valid update from its neighbor. The default route is only generated when the **gendefault** option is enabled.

importdefault

Enables GateD to accept the default route (0.0.0.0) if it is included in a received EGP update. If not specified, the default route contained in an EGP update is ignored. For efficiency, some networks have external routers announce a default route to avoid sending large EGP update packets.

exportdefault

Enables GateD to include the default route (0.0.0.0) in EGP updates sent to this EGP neighbor. This allows the system to advertise the default route via EGP. Normally, a default route is not included in EGP updates.

gateway *gateway*

If a network is not shared with a neighbor, **gateway** specifies a router on an attached network to be used as the next hop router for routes received from this neighbor. This option is only rarely used.

lcladdr *local_address*

Specifies the address to be used on the local end of the connection with the neighbor. The local address must be on an interface which is shared with the neighbor or with the neighbor's *gateway* when the **gateway** parameter is used. A session will only be opened when an interface with the appropriate local address (through which the neighbor or gateway address is directly reachable) is operating.

sourcenet *network*

Specifies the network queried in the EGP Poll packets. By default, this is the network shared with neighbors address specified. If there is no network shared with the neighbor, one of the networks the neighbor is attached to should be specified. This parameter can also be used to specify a network shared with the neighbor other than the one on which the EGP packets are sent. This parameter is normally not needed.

p1 *time*
minhello *time*

Sets the minimum acceptable interval between the transmission of EGP *HELLO* packets. The default hello interval is 30 seconds. If the neighbor fails to respond to three hello packets, GateD stops trying to acquire the neighbor. Setting a larger interval gives the neighbor a better chance to respond. **minhello** is an alias for the **P1** value defined in the EGP specification.

p2 *time*
minpoll *time*

Sets the time interval between polls to the neighbor. The default is 120 seconds. If three polls are sent without a response, the neighbor is declared "down" and all routes learned from that neighbor are removed from the routing database. A longer polling interval supports a more stable routing database but is not as responsive to routing changes. **minpoll** is an alias for the **P2** value defined in the EGP specification.

ttl *ttl*

By default, GateD sets the IP TTL for local neighbors to 1 and the TTL for non-local neighbors to 255. This option is provided when attempting to communicate with improperly-functioning routers that ignore packets sent with a TTL of one.

traceoptions *trace_options*

Specifies the tracing options for this EGP neighbor. By default, these are inherited from group or EGP global trace options. (See the Trace Statements (page 2-9) and the EGP-specific tracing options below.)

Tracing options

The **state** and **policy** options work with EGP.

Packet tracing options (which may be modified with `detail`, `send` and `recv`):

packets
All EGP packets

hello
EGP HELLO/I-HEARD-U packets which are used to determine neighbor reachability.

acquire
EGP ACQUIRE/CEASE packets which are used to initiate and terminate EGP sessions.

update
EGP POLL/UPDATE packets which are used to request and receive reachability updates.

The BGP protocol

The Border Gateway Protocol (BGP) is an exterior routing protocol used for exchanging routing information between autonomous systems. BGP is used for exchange of routing information between multiple transit autonomous systems as well as between transit and stub autonomous systems. BGP is related to EGP, but has more capability, greater flexibility, and less required bandwidth. BGP uses *path attributes* to provide more information about each route, and in particular to maintain an *AS path*, which includes the AS number of each autonomous system the route has transited, providing information sufficient to prevent routing loops in an arbitrary topology. Path attributes may also be used to distinguish between groups of routes to determine administrative preferences, allowing greater flexibility in determining route preference to achieve a variety of administrative ends.

BGP supports two basic types of sessions between neighbors, internal (sometimes referred to as IBGP) and external. Internal sessions are run between routers in the same autonomous system, while external sessions run between routers in different autonomous systems. When sending routes to an external peer, the local AS number is prepended to the AS path. This means that routes received from an external peer are guaranteed to have the AS number of that peer at the start of the path. In general, routes received from an internal neighbor will not have the local AS number prepended to the AS path, and hence will have the same AS path that the route had when the originating internal neighbor received the route from an external peer. Routes with no AS numbers in the path may be legitimately received from internal neighbors. These routes should be considered internal to the receiver's own AS.

The BGP implementation supports three versions of the BGP protocol, versions 2, 3 and 4. BGP versions 2 and 3 are quite similar in capability and function. They will only propagate classed network routes, and the AS path is a simple array of AS numbers. BGP 4 will propagate fully general address-and-mask routes, and the AS path structure can represent the results of aggregating dissimilar routes.

External BGP sessions may or may not include a single metric, which BGP calls the *Multi-Exit Discriminator* (MED), among the path attributes. For BGP versions 2 and 3, this metric is a 16-bit unsigned integer. For BGP version 4, it is a 32-bit unsigned integer. Smaller values of the Multi-Exit Discriminator are preferred. Currently this metric is only used to break ties between routes with equal preference from the same neighboring AS.

Internal BGP sessions carry at least one metric in the path attributes, which BGP calls the *LocalPref*. The range of LocalPref is identical to the range of the MED. For BGP versions 2 and 3, a route is preferred if its value for LocalPref is smaller. For BGP version 4, a route is preferred if its value for this metric is larger. BGP version 4 internal sessions may optionally include a second metric, the Multi-Exit Discriminator, carried in from external sessions. The use of these metrics is dependent on the type of internal protocol processing which is specified.

BGP collapses routes with similar path attributes into a single update for advertisement. Routes that are received in a single update will be re-advertised in a single update. The churn caused by the loss of a neighbor will be minimized and the initial advertisement sent during peer establishment will be maximally compressed. BGP does not read information from the kernel message-by-message, but fills the input buffer. It processes all complete messages in the buffer before reading again. BGP also does multiple reads to clear all incoming data queued on the socket. This feature may cause other protocols to be blocked for prolonged intervals by a busy peer connection.

All unreachable messages are collected into a single message and sent prior to reachable routes during a flash update. For these unreachable announcements, the next hop is set to the local address on the connection, no metric is sent, and the path origin is set to incomplete. On external connections the AS path in unreachable announcements is set to the local AS. On internal connections, the AS path is set to length zero.

The BGP implementation expects external peers to be directly attached to a shared subnet, and expects those peers to advertise next hops which are host addresses on that subnet (though this constraint can be relaxed by configuration for testing). For groups of internal peers, however, there are several alternatives which may be selected from by specifying the group type and route reflection options. Type **internal** groups expect all peers to be directly attached to a shared subnet so that, like external peers, the next hops received in BGP advertisements may be used directly for forwarding. Type **routing** groups instead will determine the immediate next hops for routes by using the next hop received with a route from a peer as a forwarding address, and by using this to look up an immediate next hop in an IGP's routes. Such groups support distant peers, but need to be informed of the IGP or IGPs whose routes they are using to determine immediate next hops.

For internal BGP group types (and for test groups), where possible, a single outgoing message is built for all group peers based on the common policy. A copy of the message is sent to every peer in the group, with possible adjustments to the next-hop field as appropriate to each peer. This minimizes the computational load of running large numbers of peers in these types of groups. BGP allows unconfigured peers to connect if an appropriate group has been configured with an **allow** clause.

Route reflection

Generally, all border routers in a single AS need to be internal peers of each other, and in fact all non-border routers frequently need to be internal peers of all border routers. While this is usually acceptable in small networks, it may lead to unacceptably large internal peer groups in large networks. To help address this problem, BGP supports *route reflection* for internal peer groups (with BGP version 4 only). When using route reflection, the rule that a router may not re-advertise routes from internal peers to other internal peers is relaxed for some routers, called *route reflectors*. A typical use of route reflection might involve a "core" backbone of fully meshed routers ("fully meshed" means all the routers in the fully meshed group peer directly with all other routers in the group), some of which act as route reflectors for routers which are not part of the core group.

Two types of route reflection are supported. By default, all routes received by the route reflector from a client are sent to all internal peers (including the client's group but not the client itself). If the *no-client-reflect* option is enabled, routes received from a route reflection client are sent only to internal peers which are not members of the client's group. In this case, the client's group must itself be fully meshed. In either case, all routes received from a non-client internal peer are sent to all route reflection clients.

Typically, a single router will act as the reflector for a set, or *cluster*, of clients. However, for redundancy two or more may also be configured to be reflectors for the same cluster. In this case, a cluster ID should be selected to identify all reflectors serving the cluster, using the **clusterid** keyword. Gratuitous use of multiple redundant reflectors is not advised, as it can lead to an increase in the memory required to store routes on the redundant reflectors' peers.

No special configuration is required on the route reflection clients. From a client's perspective, a route reflector is simply a normal IBGP peer. Any BGP version 4 speaker should be able to be a reflector client. (Note however that GateD versions 3.5B3 and earlier, and 3.6A1 and earlier, contain a bug which prevents them from acting as route reflection clients.)

Readers are referred to the route reflection specification document (RFC 1966 as of this writing) for further details.

Confederations

In addition to improvements in routing policy control, current techniques for deploying BGP among speakers in the same autonomous system generally require a full mesh of TCP connections among all speakers for the purpose of exchanging exterior routing information. In autonomous systems the number of intra-domain connections that need to be maintained by each border router can become significant. While this is usually acceptable in small networks, it may lead to unacceptably large internal peer groups in large networks. To help address this problem, BGP supports confederations for internal peer groups (with BGP version 4 only). It may be useful to subdivide an autonomous system into smaller domains for purposes of controlling routing policy via information contained in the BGP AS_PATH attribute (similar to EBGP). For example, one may chose to consider all BGP speakers in a geographic region as a single entity (confederation).

Subdividing a large autonomous system allows a significant reduction in the total number of intra-domain BGP connections as the connectivity requirements simplify to the model used for inter-domain connections.

There is usually no need to expose the internal topology of this divided autonomous system, which means it is possible to regard a collection of autonomous systems under a common administration as a single entity or autonomous system when viewed from outside the confines of the confederation of autonomous systems itself.

Operationally, a member of a BGP confederation will use its confederation identifier in all transactions with peers that are not members of its confederation. This confederation identifier is considered to be the "externally visible" AS number, and this number is used in OPEN messages and advertised in the AS_PATH attribute.

A member of a BGP confederation will use its routing domain identifier (the internally visible AS number) in all transactions with peers that are members of the same confederation as the given router.

A BGP speaker receiving an AS_PATH attribute containing a confederation ID matching its own confederation shall treat the path in the same fashion as if it had received a path containing its own AS number.

Thus, BGP peering sessions as the confederation border are analogous to external BGP transactions within an AS; and BGP peering sessions within the confederation are analogous to internal BGP transactions.

Readers are referred to the confederation specification document (rfc1966 as of this writing) for further details.

Communities

The Communities attribute allows the administrator of a Routing Domain to tag groups of routes with a community tag. The tag consists of 2 octets of Autonomous System (AS) and 2 octets of Community ID. The Community attribute is passed from routing domain to routing domain to maintain the grouping of these routes. A set of routes may have more than one community tag in its Community attribute.

Communities import and export policy is configured using the **aspath-opt** clause (or **mod-aspath** clause) to the group, import and export statements.

Please refer to the Communities specification and its accompanying usage documents (RFC1997 and RFC1998 as of this writing) for further details on BGP Communities.

Subgroups: differing policy for peers in the same AS/RDI

The **subgroups** variable is used to designate different sub-groups of BGP peers within the same AS (designated by the **group** statement). It works in conjunction with the subgroup identifier on the import and/or export policy configuration using the export or import clause. Essentially, the designation of the mathematical group (the group can have, at minimum, a membership of one element [peer statement]) allows for the extensibility and flexibility for "policy per peer."

Multi-exit discriminator

The Multi Exit Discriminator, or MED, allows the administrator of a Routing Domain to choose between various exits from a neighboring AS. This attribute is used only for decision making in choosing the best route to the neighboring AS. If all the other factors for a path to a given AS are equal, the path with the lower MED value takes preference over other paths.

This attribute is not propagated to other neighboring ASs. However, this attribute may be propagated to other BGP speakers within the same AS.

The MED attribute, for BGP version 4, is a four-octet unsigned integer.

MED is originated using the **metricout** option of the export, group and/or peer statement. It is imported using the **med** keyword on the BGP group statement. A MED can also be created from IGP metrics when exporting one protocol into another (e.g.a redistribution of OSPF into BGP).

Weighted route dampening

The basic idea of weighted route dampening is to treat routes that are being announced and withdrawn (flapping) at a rapid rate as unreachable.

If a route flaps at a low rate, it should not be suppressed at all, or suppressed only for a brief period of time. With weighted route dampening, the suppression of a route or routes occurs in a manner that adapts to the frequency and duration that a particular route appears to be flapping. The more a route flaps during a period of time, the longer it will be suppressed. The adaptive characteristics of weighted route dampening are controlled by a few configurable parameters.

For syntax and configuration information, refer to the Weighted Route Dampening statement (page 2-65).

Local_Pref

Routes propagated by IBGP must include a Local_Pref attribute. Local_Pref may be used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. Unless the **localpreference** option has been set on the Import or Export statements, BGP sends the Local_Pref path attribute as 100.

GateD always uses the received Local_Pref to select between BGP routes. BGP routes with a larger Local_Pref are preferred. The range of values for local preference are 0 - 2**31.

Note: All routers in the same network which are running GateD and participating in IBGP should use **localpreference** uniformly. That is, if one router has **localpreference** set, all should set it, and all should use the same value.

Route selection

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors. GateD uses the following criteria, in order, to select the best path. If routes are equal at a given point in the selection process, then the next criterion is applied to break the tie.

1. Configured Policy - The route with smallest preference, as determined by the policy defined in `gated.conf`.
2. Local_Pref - The route with the highest BGP local preference.
3. Shortest AS Path - The route with the fewest ASs listed in its AS Path.
4. Origin IGP < EGP < Incomplete - The route with an AS path origin of IGP is preferred. Next in precedence is the route with AS path origin of EGP. Least preferred is an AS path that is incomplete.
5. MED (if not ignored) - The route with the best Multi-Exit Discriminator is preferred. MEDs are only compared between routes which were received from the same neighbor AS. (So this test is only applied if the local AS has two or more connections to a given neighbor AS.)
6. Shortest IGP distance - The route whose NEXT_HOP is closer (with respect to the IGP distance) is preferred.
7. Source IGP < EBGP < IBGP - Prefer first the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.
8. Lowest Router ID - The route whose next hop IP address is numerically lowest.

The BGP statement

```
bgp yes | no | on | off
{
    protocol-precedence precedence ;
    allow bad community;          \
    defaultmetric metric ;
    traceoptions trace_options ;
    [ clusterid host ; ]
 [ group type
         ( external peeras autonomous_system
                [ ignorefirstashop ]  [ subgroup integer ]
                [ med ] )

       | ( internal peeras autonomous_system
                [ ignorefirstashop ]
                [ lcladdr local_address ]
                [ outdelay time ]
                [ metricout metric ]
                [ reflector-client [ no-client-reflect ] ]
                [ subgroup integer ]

      | ( routing peeras autonomous_system proto proto_list
                interface interface_list
                [ ignorefirstashop ]
                [ lcladdr local_address ]
                [ outdelay time ]
                [ metricout metric ]
                [ reflector-client [ no-client-reflect ] ]
                [ subgroup integer ]

      | ( confed peeras autonomous_system proto proto_list
                interface interface_list
                [ ignorefirstashop ]
                [ lcladdr local_address ]
                [ outdelay time ]
                [ metricout metric ]
                [ reflector-client [ no-client-reflect ] ]
                [ subgroup integer ]

      | ( test peeras autonomous_system )  ]
        [ aspath-opt ]
        {
          [  allow {
                network
                network mask mask
                network ( masklen | / ) number
                all
                host host  ]
```

```
            } ;
            peer host
                [ metricout metric ]
                [ localas autonomous_system ]
                [ ignorefirstashop ]
                [ nogendefault ]
                [ gateway gateway ]
                [ nexthopself ]
                [ protocol-precedence precedence ]
                [ preference preference ]
                [ lcladdr local_address ]
                [ holdtime time ]
                [ version number ]
                [ passive ]
                [ sendbuffer number ]
                [ recvbuffer number ]
                [ outdelay time ]
                [ keep [ all | none ] ]
                [ show-warnings ]
                [ noaggregatorid ]
                [ keepalivesalways ]
                [ v3asloopokay ]
                [ nov4asloop ]
                [ ascount count ]
                [ throttle count ]
                [ allow bad routerid ]
                [ logupdown ]
                [ ttl ttl ]
                [ traceoptions trace_options ]
                ;
        } ;
    } ;
```

The **bgp** statement enables or disables BGP. By default, BGP is disabled. The default metric for announcing routes via BGP is no metric.

protocol-precedence *precedence*

Sets the global preference for BGP incoming routes. The default precedence is 170. This precedence may be overridden by a precedence specified on the **group** or **peer** statement, or by import policy.

allow bad community

BGP communities are specified in RFC 1997 as being (logically) a 16-bit AS number followed by an arbitrary 16-bit integer; we refer to these as the "AS part" and the "tag part" respectively. The specification explicitly reserves communities with AS part 0 or 65535 for use as well-known communities or other future uses.

Accordingly, GateD ordinarily does not permit use of 0 or 65535 in the AS part of a community. However, some BGP implementations have been found to allow this behavior. In order to permit interoperation with such implementations, the command **allow bad community** may be used within the BGP clause. The preferred solution is to configure the other routers in use to conform to RFC 1997 by using a valid AS number (normally, an AS number assigned to you by the InterNIC) in the AS part.

On a router which configures communities as 32-bit integers rather than as an AS part and a tag part, the reserved communities are 0 through 65535 and 4294901760 through 4294967295. Usage of these communities should be avoided. Any other community (65536 through 4294901759) is legal, though it is advisable to use one's own AS number in the AS part (for example, communities 80871424 through 80936959 have AS 1234 in the AS part).

defaultmetric *metric*

Defines the metric used when advertising routes via BGP. If not specified, no metric is propagated. This metric may be overridden by a metric specified on the neighbor or group statements or in export policy.

traceoptions *trace_options*

Specifies the tracing options for BGP. By default these are inherited from the global trace options. These values may be overridden on a group or neighbor basis. (See the Trace Statements (page 2-9) and the BGP-specific tracing options below.)

clusterid *host*

Specifies the route reflection cluster ID for BGP. The cluster ID defaults to be the same as the router ID. If a router is to be a route reflector, then a single cluster ID should be selected and configured on all route reflectors in the cluster. The only constraints on the choice of cluster ID is that (a) IDs of clusters within an AS must be unique within that AS, and (b) the cluster ID must not be 0.0.0.0. Choosing the cluster ID to be the router ID of one router in the cluster will always fulfill these criteria. If there is only one route reflector in the cluster, the **clusterid** setting may be omitted, as the default will suffice.

Groups

BGP peers are grouped by type and the autonomous system of the peers. Any number of groups may be specified, but each must have a unique combination of type, peer autonomous system and **aspath-opt** options. There are four possible group types:

group type external peeras *autonomous_system* [**med**] [**ignorefirstashop**]
[**subgroup** *integer*] [*aspath-opt*]

In the classic external BGP group, full policy checking is applied to all incoming and outgoing advertisements. The external neighbors must be directly reachable through one of the machine's local interfaces. The next hop transmitted is computed with respect to the shared interface.

If the **gateway** parameter is present on peer statement, EBGP peers can be on different networks. Refer to the **gateway** keyword statement described later in this BGP section (page 2-60).

med

By default, any metric (Multi_Exit_Disc, or MED) received on a BGP connection is ignored. If it is desired to use MEDs in routing computations, the **med** option must be specified on the group. By default, MEDs are not sent on external connections. To send MEDs, use the "**metric**" option of the export statement or the "**metricout**" peer/group parameter

ignorefirstashop

Some routers, known as *Route Servers*, are capable of propagating routes without appending their own AS to the AS Path. By default, GateD will drop such routes. Specifying **ignorefirstashop** on either the group statement or peer clause disables this feature. This option should only be used if it is positively known that the peer is a route server and not a normal router.

subgroup *integer*

The subgroup variable is to designate different sub-groups of BGP peers within the same AS (designated by the group statement). It works in conjunction with the subgroup identifier on the import and/or export policy configuration using the export or import clause. Essentially, the designation of the mathematical group (the group can have at minimum a membership of one element [peer statement]) allows for the extensibility and flexibility for "policy per peer."

aspath-opt

Originate the specified AS path attributes. If the attributes are already present they may be augmented (as with communities) or possibly replaced (other attributes which may be supported in the future).

group type routing peeras *autonomous_system* **proto** *proto* [**interface** *interface_list*]
[**reflector-client** [**no-client-reflect**]] [**ignorefirstashop**] [**lcladdr** *local_address*]
[**outdelay** *time*] [**metricout** *metric*] [**subgroup** *integer*] [*aspath-opt*]

An internal group which uses the routes of an interior protocol to resolve forwarding addresses. A **type routing** group propagates external routes between routers which are not directly connected, and computes immediate next hops for these routes by using the BGP next hop which arrived with the route as a forwarding address to be resolved via an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, while the IGP is expected to only carry AS internal routes, and the latter is used to find immediate next hops for the former.

The *proto* names the interior protocol to be used to resolve BGP route next hops, and may be the name of one or more IGPs in the configuration, including "static", "ISIS", "OSPF", "direct", "BGP", and "RIP". "All" may also be used, though its use is discouraged. By default, the next hop in BGP routes advertised to type routing peers will be the original nexthop that was specified in the arriving BGP UPDATE packet. The *interface_list* can optionally provide a list of interfaces whose routes are carried via the IGP for which third-party next hops may be used instead.

lcladdr, **outdelay**, and **metricout** must be set in the group statement, not on a per-peer basis, for the group types Internal and Routing. If these options are set on the peer statement, they must equal the values set on the corresponding group statement. **lcladdr** is required if you want to use loopback alias as `router_id`.

The **reflector-client** option specifies that GateD will act as a route reflector for this group. All routes received from any group member will be sent to all other internal neighbors, and all routes received from any other internal neighbors will be sent to the reflector clients. Since the route reflector forwards routes in this way, the reflector-client group need not be fully meshed. If the **no-client-reflect** option is specified, routes received from reflector clients will only be sent to internal neighbors which are *not* in the same group as the sending reflector client. In this case, the reflector-client group should be fully meshed. In all cases, routes received from normal internal peers will be sent to all reflector clients.

Note that it is necessary to export routes from the local AS *into* the local AS when acting as a route reflector. For example, suppose that the local AS number is 2.
An export statement like the following would suffice to make reflection work right:

```
export proto bgp as 2 {
      proto bgp as 2 {all;}; # for reflection
      # other exports
};
```

If the cluster ID is changed and GateD is reconfigured with a SIGHUP, all BGP sessions with reflector clients will be dropped and restarted.

subgroup *integer*

The subgroup variable is to designate different sub-groups of BGP peers within the same AS (designated by the group statement). It works in conjunction with the subgroup identifier on the import and/or export policy configuration using the export or import clause. Essentially, the designation of the mathematical group (the group can have at minimum a membership of one element [peer statement]) allows for the extensibility and flexibility for "policy per peer."

group type confed peeras*autonomous_system* **proto** *proto_list* **interface** *interface_list*
[**reflector-client** [**no-client-reflect**]] [**ignorefirstashop**] [**lcladdr** *local_address*]
[**outdelay** *time*] [**metricout** *metric*] [**subgroup** *integer*] [*aspath-opt*]

An internal group which uses the routes of an interior protocol to resolve forwarding addresses. A **type confed** group propagates external routes between routers which are not directly connected, and computes immediate next hops for these routes by using the BGP next hop which arrived with the route as a forwarding address to be resolved via an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, while the IGP is expected to only carry AS internal routes, and the latter is used to find immediate next hops for the former.

The *proto_list* names the interior protocol to be used to resolve BGP route next hops, and may be the name of one or more IGPs in the configuration, including "static", "ISIS", "OSPF", "direct", "BGP", and "RIP". "All" may also be used, though its use is discouraged. By default the next hop in BGP routes advertised to type routing peers will be the original nexthop that was specified in the arriving BGP UPDATE packet. The *interface_list* can optionally provide a list of interfaces whose routes are carried via the IGP for which third-party next hops may be used instead.

The **lcladdr**, **outdelay**, and **metricout** must be set in the group statement, not on a per-peer basis, for the group types Internal and Routing. If these options are set on the peer statement, they must equal the values set on the corresponding group statement.

The **reflector-client** option specifies that gated will act as a route reflector for this group. All routes received from any group member will be sent to all other internal neighbors, and all routes received from any other internal neighbors will be sent to the reflector clients. Since the route reflector forwards routes in this way, the reflector-client group need not be fully meshed. If the **no-client-reflect** option is specified, routes received from reflector clients will only be sent to internal neighbors which are not in the same group as the sending reflector client. In this case the reflector-client group should be fully meshed. In all cases, routes received from normal internal peers will be sent to all reflector clients. Note that it is necessary to export routes from the local AS into the local AS when acting as a route reflector.

For example, suppose that the local AS number is 2. An export statement like the following would suffice to make reflection work right:

```
export proto bgp as 2 {
      proto bgp as 2 {all;}; # for reflection
      # other exports
};
```

If the cluster ID is changed and **gated** is reconfigured with a SIGHUP, all BGP sessions with reflector clients will be dropped and restarted.

subgroup *integer*

The subgroup variable is to designate different sub-groups of BGP peers within the same AS (designated by the group statement). It works in conjunction with the subgroup identifier on the import and/or export policy configuration using the export or import clause. Essentially, the designation of the mathematical group (the group can have at minimum a membership of one element [peer statement]) allows for the extensibility and flexibility for "policy per peer."

group type internal peeras *autonomous_system* [**reflector-client** [**no-client-reflect**]] [**ignorefirstashop**] [**lcladdr** *local_address*] [**outdelay** *time*] [**metricout** *metric*]

[**subgroup** *integer*] [*aspath-opt*]

An internal group operating where there is no IP-level IGP, for example, an SMDS network or MILNET. All neighbors in this group are required to be directly reachable via a single interface. All next-hop information is computed with respect to this interface. Import and export policy may be applied to group advertisements. Routes received from external BGP or EGP neighbors are by default readvertised with the received metric.

The **lcladdr**, **outdelay**, **metricout**, **reflector-client** and **no-client-reflect** options are described under the **routing** group description. **lcladdr** is required if you want to use loopback alias as `router_id`.

subgroup *integer*

The subgroup variable is to designate different sub-groups of BGP peers within the same AS (designated by the group statement). It works in conjunction with the subgroup identifier on the import and/or export policy configuration using the export or import clause. Essentially, the designation of the mathematical group (the group can have at minimum a membership of one element [peer statement]) allows for the extensibility and flexibility for "policy per peer."

group type test peeras *autonomous_system*

An extension to external BGP which implements a fixed policy using test peers. Fixed policy and *special case* code make test peers relatively inexpensive to maintain. Test peers do not need to be on a directly-attached network. If GateD and the peer are on the same (directly-attached) subnet, the advertised next hop is computed with respect to that network, otherwise, the next hop is the local machine's current next hop. All routing information advertised by and received from a test peer is discarded, and all BGP-advertiseable routes are sent back to the test peer. Metrics from EGP- and BGP-derived routes are forwarded in the advertisement; otherwise, no metric is included.

Group parameters

The BGP statement has **group** clauses and **peer** subclauses. Any number of **peer** subclauses may be specified within a group. A **group** clause usually defines default parameters for a group of peers. These parameters apply to all subsidiary **peer** subclauses. Any parameters from the **peer** subclause may be specified on the **group** clause to provide defaults for the whole group (which may be overridden for individual peers).

Specifying peers

Within a group, BGP peers may be configured in one of two ways. They may be explicitly configured with a **peer** statement, or implicitly configured with the **allow** statement. Both are described here.

allow

The **allow** clause allows **peer** connections from any addresses in the specified range of network and mask pairs. All parameters for these peers must be configured on the group clause. The internal peer structures are created when an incoming open request is received and destroyed when the connection is broken. For more detail on specifying the network/mask pairs, see the section on Route Filtering (page 2-82).

peer *host*

A **peer** clause configures an individual peer. Each peer inherits all parameters specified on a group as defaults. Many defaults may be overridden by parameters explicitly specified on the peer subclause.

Within each **group** clause, individual peers can be specified or a group of *potential* peers can be specified using **allow**. **allow** is used to specify a set of address masks. If GateD receives a BGP connection request from any address in the set specified, it will accept it and set up a peer relationship.

Peer parameters

The BGP peer subclause allows the following parameters, which can also be specified on the **group** clause. All are optional.

metricout *metric*

If specified, this metric may be used on all routes sent to the specified peer(s). The metric hierarchy is as follows, starting from the most preferred:
1) The metric specified by export policy.
2) Peer-level metricout.
3) Group-level metricout
4) Default metric.

For group types Internal and Routing, set this option on the **group** clause instead of on the **peer** clause.

localas *autonomous_system*

Identifies the autonomous system which GateD is representing to this group of peers. The default is that which has been set globally in the **autonomoussystem** statement.

nogendefault

Prevents GateD from generating a default route when BGP receives a valid update from its neighbor. The default route is only generated when the **gendefault** option is enabled.

ignorefirstashop

Disable dropping of routes from peers which do not insert their own AS number into the AS Path. This option should only be used if it is positively known that the peer is a route server and not a normal router.

gateway *gateway*

If a network is not shared with a peer, **gateway** specifies a router on an attached network to be used as the next hop router for routes received from this neighbor. The **gateway** parameter may also be used to specify a next hop for peers which are on shared networks. This might be used to ensure that third-party next hops are never accepted from a given peer, by specifying that peer's address as its own gateway. This parameter is not needed in most cases.

nexthopself

Causes the next hop in route advertisements set to this peer or group of peers to be set to our own router's address, even if it would normally be possible to send a third-party next hop. Use of this option may cause inefficient routes to be followed, but it may be needed in some cases to deal with broken bridged interconnect media (in cases where the routers on the “shared” medium do not really have full connectivity to each other), or in broken political situations.

precedence *precedence*

See above.

preference *preference*

Specifies the preference used for routes learned from these peers. This can differ from the default BGP preference set in the **bgp** statement, so that GateD can prefer routes from one peer, or group of peers, over others. This preference may be explicitly overridden by import policy. The default value is 0.

lcladdr *local_address*

Specifies the address to be used on the local end of the TCP connection with the peer. For *external* peers the local address must be on an interface which is shared with the peer or with the peer's *gateway* when the **gateway** parameter is used. A session with an external peer will only be opened when an interface with the appropriate local address (through which the peer or gateway address is directly reachable) is operating. For other types of peers, a peer session will be maintained when any interface with the specified local address is operating. In either case, incoming connections will only be recognized as matching a configured peer if they are addressed to the configured local address. For group types Internal and Routing, set this option on the **group** clause. For group type Routing, it is advisable to set the **lcladdr** to a non-physical interface, such as a *loopback* interface.

holdtime *time*

Specifies the BGP holdtime value to use when negotiating the connection with this peer, in seconds. If GateD does not receive a keepalive, update, or notification message within the number of seconds specified in the Hold Time field of the BGP Open message, then the BGP connection will be closed. The value must be at least 3 minutes, or 180 seconds.

version *version*

Specifies the version of the BGP protocol to use with this peer. If not specified, the highest supported version is used first and version negotiation is attempted. If it is specified, only the specified version will be offered during negotiation. Currently supported versions are 2, 3 and 4.

passive

If this option is used, GateD will never try to open a BGP connection with this peer (or group). Instead, it will wait for the peer to initiate a connection. This option was introduced to handle a problem in BGP3 and earlier, where two peers might both attempt to initiate a connection at the same time. This problem is fixed in the BGP4 protocol, so passive is not needed with BGP4 sessions. Note that if it is applied to both sides of a peering session, passive will prevent the session from ever being established. For this reason, and because it is generally not needed, the use of passive is discouraged.

sendbuffer *buffer_size*
recvbuffer *buffer_size*

Control the amount of send and receive buffering asked of the kernel. The maximum supported is 65535 bytes although many kernels have a lower limit. By default, GateD configures the maximum supported. These parameters are not needed on normally-functioning systems.

outdelay *time*

Used to dampen route fluctuations. **outdelay** is the number of seconds a route must be present in the GateD routing database before it is exported to BGP. The default value for each is 0, meaning that these features are disabled. For group types Internal and Routing, set this option on the **group** clause.

keep all

Used to retain routes learned from a peer even if the routes' AS paths contain one of our exported AS numbers.

show-warnings

Causes GateD to issue warning messages when receiving questionable BGP updates such as duplicate routes and/or deletions of non-existing routes. Normally these events are silently ignored.

noaggregatorid

Causes GateD to specify the routerid in the aggregator attribute as zero (instead of its routerid) in order to prevent different routers in an AS from creating aggregate routes with different AS paths.

keepalivesalways

Causes GateD to always send keepalives, even when an update could have correctly substituted for one. This allows interoperability with routers that do not completely obey the protocol specifications on this point.

v3asloopokay

By default GateD will not advertise routes whose AS path is looped (i.e. with an AS appearing more than once in the path) to version 3 external peers. Setting this flag removes this constraint. Ignored when set on internal groups or peers.

nov4asloop

Prevents routes with looped AS paths from being advertised to version 4 external peers. This can be useful to avoid advertising such routes to peer which would incorrectly forward the routes on to version 3 neighbors.

ascount *count*

count is the number of times we will insert our own AS number when we send the AS path to an external neighbor. Legal values are 1..25, inclusive, the default is 1.

Higher values are typically used to bias upstream neighbors' route selection. (All things being equal, most routers will prefer to use routes with shorter AS Paths. Using **ascount**, the AS Path we send can be artificially lengthened.) Note that **ascount** supersedes the **nov4asloop** option -- regardless of whether **nov4asloop** is set, we will still send multiple copies of our own AS if the ascount option is set to something greater than one. Also, note that if the value of ascount is changed and GateD is reconfigured, routes will **not** be sent to reflect the new setting. If this is desired, it will be necessary to restart the peer session (by commenting out the peer or group, reconfiguring, and then uncommenting and reconfiguring again, or by restarting GateD).

throttle *count*

Limit the number of UPDATE packets to *count* per second. It was found that non-GRF routers, when heavily overburdened; would "timeout" BGP peering sessions because they could not keep up with the processing of packets forwarded by the GRF.

allow bad routerid

The BGP specification specifically requires that a BGP router choose a reasonable value (one of its IP addresses) as its router ID. If a BGP OPEN message is received with an unreasonable router ID, the specification requires that an error message be sent, and the BGP connection closed (see RFC 1771, section 6.2). GateD obeys this requirement.

Apparently, some non-GateD BGP implementations sometimes decide to send 0.0.0.0 as their router ID. This is bad, not only because it violates the BGP specification, but because some elements of the BGP protocol assume that all router IDs are globally unique. Proper router ID assignment assures this, but if two routers in the same AS go insane and are allowed to send router ID 0.0.0.0, routing problems will result, especially if route reflection is in use. In particular, two routers using identical router IDs (whether 0.0.0.0 or otherwise) will not have each other's routes reflected to them.

Despite this, GateD provides a way to disable router ID sanity checking, through the use of the **allow bad routerid** peer option. Use of this option is STRONGLY DISCOURAGED, and is provided *only* as a means of allowing interoperation with a faulty router for a short period of time until the faulty router can be fixed or replaced.

logupdown

Causes a message to be logged via the **syslog** mechanism whenever a BGP peer enters or leaves the **ESTABLISHED** state.

ttl *ttl*

By default, GateD sets the IP TTL for local peers to *one* and the TTL for non-local peers to 255. This option mainly is provided when attempting to communicate with improperly functioning routers that ignore packets sent with a TTL of one. Not all kernels allow the TTL to be specified for TCP connections.

traceoptions *trace_options*

Specifies the tracing options for this BGP neighbor. By default these are inherited from group or BGP global trace options. (See the Trace Statements (page 2-9) and the BGP specific tracing options below.)

Tracing options

Note that the **state** option works with BGP, but does not provide true state transition information.

Packet tracing options (which may be modified with **detail**, **send**, and **recv**) are:

packets

All BGP packets

open

BGP **OPEN** packets which are used to establish a peer relationship.

update

BGP **UPDATE** packets which are used to pass network reachability information.

keepalive

BGP **KEEPALIVE** packets which are used to verify peer reachability.

all

Additional useful information, including additions/changes/deletions to the GateD routing table.

Limitations

If an interface goes down and comes back up and there is policy associated with that interface, GateD must be restarted because the interface policy is not recovered.

Weighted route dampening statement

Currently, only routes learned via BGP are subject to weighted route dampening although no protocols announce suppressed routes.

Note: The weighted route dampening configuration statement is not within the BGP statement, but is a separate and distinct configuration conceptually, much like interface or kernel statements.

The syntax for weighted route damping in GateD is:

```
dampen-flap {
[suppress-above metric;
reuse-below metric;
max-flap metric;
unreach-decay time;
reach-decay  time;
keep-history  time; ]
};
```

suppress-above *metric*

This is the value of the instability metric at which route suppression will take place. A route will not be installed in the forwarding information base (FIB), or announced even if it is reachable during the period that it is suppressed.

reuse-below *metric*

This is the value of the instability metric at which a suppressed route will become unsuppressed if it is reachable but currently suppressed. The value assigned to **reuse-below** must be less than **suppress-above.**

max-flap *metric*

This is the upper limit of the instability metric. This value must be greater than the larger of 1 and **suppress_above**.

reach-decay *time*

This value specifies the time desired for the instability metric value to reach one-half of its current value when the route is reachable. This half-life value determines the rate at which the metric value is decayed. A smaller half-life value will make a suppressed route reusable sooner than a larger value. Specify in seconds.

unreach-decay *time*

This value acts the same as **reach-decay** except that it specifies the rate at which the instability metric is decayed when a route is unreachable. It should have a value greater than or equal to **reach-decay**. Specify in seconds.

keep-history *time*

This value specifies the period over which the route flapping history is to be maintained for a given route. The size of the configuration arrays described below is directly affected by this value. Specify in seconds.

When only **dampen-flap {};** is specified in the route dampening statement, then the following default values are currently used:

- **suppress-above** = 3.0;
- **reuse-below** = 2.0;
- **max-flap** = 16.0;
- **unreach-decay** = 900;
- **reach-decay** = 300;
- **keep-history** = 1800;

The ICMP statement

On systems without the BSD routing socket, GateD listens to ICMP messages received by the system. Currently GateD only does processing on ICMP redirect packets, but more functionality may be added in the future, such as support for the router discovery messages. Processing of ICMP redirect messages is handled by the `redirect` statement.

Currently, the only reason to specify the `icmp` statement is to be able to trace the ICMP messages that GateD receives.

ICMP Statement

```
icmp {
    traceoptions trace_options ;
}
```

traceoptions *trace_options* ;
Specifies the tracing options for ICMP. (See the Trace Statements (page 2-9) and the ICMP-specific tracing options below.)

Tracing options

Packet tracing options (which may be modified with `detail` and `recv`):

packets
All ICMP packets received.

redirect
Only ICMP `REDIRECT` packets received.

routerdiscovery
Only ICMP `ROUTER DISCOVERY` packets received.

info
Only ICMP informational packets, which include mask request/response, info request/response, echo request/response and time stamp request/response.

error
Only ICMP error packets, which include time exceeded, parameter problem, unreachable and source quench.

The Router Discovery Protocol

The Router Discovery Protocol is an IETF standard protocol, RFC 1256, used to inform hosts of the existence of routers. It is intended to be used instead of having hosts *wiretap* routing protocols such as RIP. It is used in place of, or in addition to, statically-configured default routes in hosts.

The protocol is split into two portions, the *server* portion which runs on routers, and the *client* portion which runs on hosts. GateD treats these much like two separate protocols, only one of which may be enabled at a time.

The Router Discovery server

The Router Discovery Server runs on routers and announces their existence to hosts. It does this by periodically multicasting or broadcasting a **Router Advertisement** to each interface on which it is enabled. These Router Advertisements contain a list of all the router's addresses on a given interface and the preference of each address for use as the default router on that interface.

Initially these Router Advertisements occur every few seconds, then fall back to every few minutes. In addition, a host may send a **Router Solicitation** to which the router will respond with a unicast Router Advertisement (unless a multicast or broadcast advertisement is due momentarily).

Each Router Advertisement contains an *Advertisement Lifetime* field indicating for how long the advertised addresses are valid. This lifetime is configured such that another Router Advertisement will be sent before the lifetime has expired. A lifetime of zero is used to indicate that one or more addresses are no longer valid.

On systems supporting IP multicasting, the Router Advertisements are, by default, sent to the all-hosts multicast address `224.0.0.1`. However, the use of broadcast may be specified. When Router Advertisements are being sent to the all-hosts multicast address, or an interface is configured for the limited-broadcast address `255.255.255.255`, all IP addresses configured on the physical interface are included in the Router Advertisement. When the Router Advertisements are being sent to a net or subnet broadcast, only the address associated with that net or subnet is included.

The Router Discovery server statement

```
routerdiscovery server yes | no | on | off [ {
    traceoptions trace_options ;
    interface interface_list
        [ minadvinterval time ] |
        [ maxadvinterval time ] |
        [ lifetime time ]
        ;
    address interface_list
        [ advertise ] | [ ignore ] |
        [ broadcast ] | [ multicast ] |
        [ ineligible ] | [ preference preference ]
        ;
} ] ;
```

traceoptions *trace_options*

Specifies the Router Discovery tracing options. (See the Trace Statements (page 2-9) and the Router Discovery specific tracing options below.)

interface *interface_list*

Specifies the parameters that apply to physical interfaces.

Note a slight difference in convention from the rest of GateD; **interface** specifies a list of physical interfaces (such as `le0`, `ef0` and `en1`), while **address** specifies a list of IP addresses.

One or more of the following parameters must be provided for the interface:

maxadvinterval *time*

The maximum time allowed between sending broadcast or multicast Router Advertisements from the interface. Must be no less than 4 and no more than 30:00 (30 minutes or 1800 seconds). The default is **10:00** (10 minutes or 600 seconds).

minadvinterval *time*

The minimum time allowed between sending unsolicited broadcast or multicast Router Advertisements from the interface. Must be no less than 3 seconds and no greater than **maxadvinterval**. The default is **0.75 * maxadvinterval**.

lifetime *time*

The lifetime of addresses in a Router Advertisement. Must be no less than **maxadvinterval** and no greater than 2:30:00 (two hours, thirty minutes or 9000 seconds). The default is **3 * maxadvinterval**.

address *interface_list*

Specifies the parameters that apply to the specified set of addresses on this physical interface.

Note a slight difference in convention from the rest of GateD; **interface** specifies a list of physical interfaces (such as `le0`, `ef0` and `en1`), while **address** specifies a list of IP addresses.

One or more of the following parameters must be provided for the address:

advertise

Specifies that the specified address(es) should be included in Router Advertisements. This is the default.

ignore

Specifies that the specified address(es) should not be included in Router Advertisements.

broadcast

Specifies that the given address(es) should be included in a broadcast Router Advertisement because this system does not support IP multicasting, or some hosts on attached network do not support IP multicasting. It is possible to mix addresses on a physical interface such that some are included in a broadcast Router Advertisement and some are included in a multicast Router Advertisement. This is the default if the router does not support IP multicasting.

multicast

Specifies that the given address(es) should only be included in a multicast Router Advertisement. If the system does not support IP multicasting the address(es) will not be included. If the system supports IP multicasting, the default is to include the address(es) in a multicast Router Advertisement if the given interface supports IP multicasting. If not, the address(es) will be included in a broadcast Router Advertisement.

preference *preference*

The degree of preference of the address(es) as a default router address, relative to other router addresses on the same subnet. This is a 32-bit, signed, two's-complement integer, with higher values meaning more preferable. Note that `hex 80000000` may only be specified as **ineligible**. The default is **0**.

ineligible

Specifies that the given address(es) will be assigned a preference of `hex 80000000` which means that it is not eligible to be the default route for any hosts.

This is useful when the address(es) should not be used as a default route, but are given as the next hop in an ICMP redirect. This allows the hosts to verify that the given addresses are up and available.

The Router Discovery client

A host listens for Router Advertisements via the all-hosts multicast address (`224.0.0.2`), if IP multicasting is available and enabled, or on the interface's broadcast address. When starting up, or when reconfigured, a host may send a few Router Solicitations to the all-routers multicast address, `224.0.0.2`, or the interface's broadcast address.

When a Router Advertisement with non-zero lifetime is received, the host installs a default route to each of the advertised addresses. If the preference is **ineligible**, or the address is not on an attached interface, the route is marked unusable but retained. If the preference is usable, the metric is set as a function of the preference such that the route with the best preference is used. If more than one address with the same preference is received, the one with the lowest IP address will be used. These default routes are not exportable to other protocols.

When a Router Advertisement with a zero lifetime is received, the host deletes all routes with next-hop addresses learned from that router. In addition, any routers learned from ICMP redirects pointing to these addresses will be deleted. The same will happen when a Router Advertisement is not received to refresh these routes before the lifetime expires.

The router discovery client statement

```
routerdiscovery client yes | no | on | off [ {
    traceoptions trace_options ;
    preference preference ;
    interface interface_list
        [ enable ] | [ disable ]
        [ multicast ]
        [ quiet ] | [ solicit ]
        ;
} ] ;
```

traceoptions *trace_options*

Specifies the tracing options for OSPF. (See the Trace Statements (page 2-9) and the OSPF specific tracing options below.)

preference *preference* ;

Specifies the preference of all Router Discovery default routes. The default is **55**.

interface *interface_list*

Specifies the parameters that apply to physical interfaces. Note a slight difference in convention from the rest of GateD; **interface** specifies just physical interfaces (such as `le0`, `ef0` and `en1`). The Router Discovery Client has no parameters that apply only to interface addresses.

enable

Specifies that Router Discovery should be performed on the specified interface(s). This is the default.

disable

Specifies that Router Discovery should not be performed on the specified interface(s).

multicast

Specifies that Router Solicitations should be multicast on the specified interface(s). If IP multicast is not available on this host and interface, no solicitation will be performed. The default is to multicast Router Solicitations if the host and interface support it, otherwise Router Solicitations are broadcast.

quiet

Specifies that no Router Solicitations will be sent on this interface, even though Router Discovery will be performed.

solicit

Specifies that initial Router Solicitations will be sent on this interface, this is default.

Tracing options

The Router Discovery Client and Server support the **state** trace flag which traces various protocol occurrences.

state

State transitions

The Router Discovery Client and Server do not directly support any packet tracing options. Tracing of router discovery packets is enabled via the ICMP Statement.

The kernel statement

While the kernel interface is not technically a routing protocol, it has many characteristics of one, and GateD handles it similarly to one. The routes GateD chooses to install in the kernel forwarding table are those that will actually be used by the kernel to forward packets.

The add, delete and change operations GateD must use to update the typical kernel forwarding table take a non-trivial amount of time. This does not present a problem for older routing protocols (RIP, EGP), which are not particularly time critical and do not easily handle very large numbers of routes anyway. The newer routing protocols (OSPF, BGP) have stricter timing requirements and are often used to process many more routes. The speed of the kernel interface becomes critical when these protocols are used.

To prevent GateD from locking up for significant periods of time installing large numbers of routes (up to a minute or more has been observed on real networks), the processing of these routes is now done in batches. The size of these batches may be controlled by the tuning parameters described below, but normally the default parameters will provide the proper functionality.

During normal shutdown processing, GateD normally deletes all the routes it has installed in the kernel forwarding table, except for those marked with **retain**. Optionally, GateD can leave all routes in the kernel forwarding table by not deleting any routes. In this case changes will be made to insure that routes with a **retain** indication are installed in the table. This is useful on systems with large numbers of routes as it prevents the need to re-install the routes when GateD restarts. This can greatly reduce the time it takes to recover from a restart.

Forwarding tables and routing tables

The table in the kernel that controls the forwarding of packets is a forwarding table, also known in ISO speak as a forwarding information base, or FIB. The table that GateD uses internally to store routing information it learns from routing protocols is a routing table, known in ISO speak as a routing information base, or RIB. The routing table is used to collect and store routes from various protocols. For each unique combination of *network* and *mask*, an *active* route is chosen, this route will be the one with the best (numerically smallest) *preference*. All the active routes are installed in the kernel forwarding table. The entries in this table are what the kernel actually uses to forward packets.

Updating the forwarding table

There are two main methods of updating the kernel FIB, the `ioctl`() interface and the routing socket interface. Their various characteristics are described here.

Updating the forwarding table with the ioctl interface

The `ioctl` interface to the forwarding table was introduced and widely distributed in BSD 4.3. This is a one-way interface, it only allows GateD to update the kernel forwarding table. It has several other limitations:

- Fixed subnet masks

  The BSD 4.3 networking code assumed that all subnets of a given network had the same subnet mask. This limitation is enforced by the kernel. The network mask is not stored in the kernel forwarding table, but determined when a packet is forwarded by searching for interfaces on the same network.

- One-way interface

  GateD is able to update the kernel forwarding table, but it is not aware of other modifications of the forwarding table. GateD is able to listen to ICMP messages and guess how the kernel has updated the forwarding table with response to ICMP redirects.
- Blind updates

  GateD is not able to detect changes to the forwarding table resulting from the use of the *route* command by the system administrator. Use of the *route* command on systems that use the `ioctl`() interface is strongly discouraged while GateD is running.
- Changes not supported

  In all known implementations, there is no change operation supported, to change a route that exists in the kernel, the route must be deleted and a new one added.

Updating the forwarding table with the routing socket interface

The *routing socket* interface to the kernel forwarding table was introduced in *BSD 4.3 Reno*, widely distributed in *BSD 4.3 Net/2* and improved in *BSD 4.4*. This interface is simply a socket, similar to a UDP socket, on which the kernel and GateD exchange messages. It has several advantages over the `ioctl`() interface:

- Variable subnet masks

  The network mask is passed to the kernel explicitly. This allows different masks to be used on subnets of the same network. It also allows routes with masks that are more general than the natural mask to be used. This is known as classless routing.
- Two-way interface

  Not only is GateD able to change the kernel forwarding table with this interface, but the kernel can also report changes to the forwarding table to GateD. The most interesting of these is an indication that a redirect has modified the kernel forwarding table; this means that GateD no longer needs to monitor ICMP messages to learn about redirects. Plus, there is an indication of whether the kernel processed the redirect, GateD can safely ignore redirect messages that the kernel did not process.
- Updates visible

  Changes to the routing table by other processes, including the *route* command are received via the routing socket. This allows GateD to insure that the kernel forwarding table is in sync with the routing table. Plus it allows the system administrator the ability to do some operations with the *route* command while GateD is running.
- Changes supported

  There is a functioning *change* message that allows routes in the kernel to be atomically changed. Some early versions of the routing socket code had bugs in the change message processing. There are compilation time and configuration time options that cause delete and add sequences to be used in lieu of change messages.
- Expandable

  New levels of kernel/GateD communications may be added by adding new message types.

Reading the forwarding table

When GateD starts up, it reads the kernel forwarding table and installs corresponding routes in the routing table. These routes are called *remnants* and are timed out after a 3-minute interval), or as soon as a more attractive route is learned. This allows forwarding to occur during the time it takes the routing protocols to start learning routes.

There are three main methods for reading the forwarding table from the kernel.

Reading forwarding table via kmem

On many systems, especially those based on BSD 4.3, GateD must have knowledge of the kernel's data structures and poke around in the kernel to read the current state of forwarding table. This method is slow and subject to error if the kernel forwarding table is updated while GateD is in the middle of reading it. This can happen if the system administrator uses the *route* command, or an ICMP redirect message is received while GateD is starting up.

Due to an oversight, some systems, such as OSF/1, which are based on BSD 4.3 Reno or later, do not have the `getkerninfo`() system call described below which allows GateD to read routes from the kernel without know about kernel internal structures. On these systems it is necessary to read the kernel radix tree from the kernel by poking around in kernel memory. This is even more error prone than reading the hash-based forwarding table.

Reading the forwarding table via getkerninfo/sysctl

Besides the routing socket, BSD 4.3 Reno introduced the `getkerninfo`() system call. This call allows a user process (of which GateD is one) to read various information from the kernel without knowledge of the kernel data structures. In the case of the forwarding table, it is returned to GateD atomically as a series of routing socket messages. This prevents the problem associated with the forwarding table changing while GateD is in the process of reading it.

BSD 4.4 changed the `getkerninfo`() interface into the `sysctl`() interface, which takes different parameters, but otherwise functions identically.

Reading the forwarding table via OS-specific methods

Some operating systems, for example SunOS 5, define their own method of reading the kernel forwarding table. The SunOS 5 version is similar in concept to the `getkerninfo`() method.

Reading the interface list

The kernel support subsystem of GateD is responsible for reading the status of the kernel's physical and protocol interfaces periodically. GateD detects changes in the interface list and notifies the protocols so they can start or stop instances or peers. The interface list is read one of two ways:

Reading the interface list with SIOCGIFCONF

On systems based on BSD 4.3, 4.3 Reno and 4.3 Net/2 the `SIOCGIFCONF ioctl` interface is used to read the kernel interface list. Using this method a list of interfaces and some basic information about them is return by the SIOCGIFCONF call. Other information must be learned by issuing other ioctls to learn the interface network mask, flags, MTU, metric, destination address (for point-to-point interfaces) and broadcast address (for broadcast capable interfaces).

GateD reads re-reads this list every 15 second looking for changes. When the routing socket is in use, it also re-reads it whenever a messages is received indicating a change in routing configuration. Receipt of a `SIGUSR2` signal also causes GateD to re-read the list. This interval may be explicitly configured in the interface configuration.

Reading the interface list with sysctl

BSD 4.4 added the ability to read the kernel interface list via the `sysctl` system call. The interface status is returned atomically as a list of routing socket messages which GateD parses for the required information.

BSD 4.4 also added routing socket messages to report interface status changes immediately. This allows GateD to react quickly to changes in interface configuration.

When this method is in use, GateD re-reads the interface list only once a minute. It also re-reads it on routing table changes indications and when a SIGUSR2 is received. This interval may be explicitly configured in the interface configuration.

Reading interface physical addresses

Later version of the `getkerninfo`() and `sysctl`() interfaces return the interface physical addresses as part of the interface information. On most systems where this information is not returned, GateD scans the kernel physical interface list for this information for interfaces with IFF_BROADCAST set, assuming that their drivers are handled the same as Ethernet drivers. On some systems, such as *SunOS 4* and *SunOS 5*, system specific interfaces are used to learn this information

The interface physical addresses are useful for IS-IS, for IP protocols, they are not currently used, but may be in the future.

Reading kernel variables

At start-up, GateD reads some special variables out of the kernel. This is usually done with the `nlist` (or `kvm_nlist`) system call, but some systems use different methods.

The variables that are read include the status of UDP *checksum creation* and *generation*, *IP forwarding* and *kernel version* (for informational purposes). On systems where the routing table is read directly from kernel memory, the root of the *hash table* or *radix tree routing table* is read. On systems where interface physical addresses are not supplied by other means, the root of the *interface list* is read.

Special route flags

The later BSD-based kernel supports the special route flags described here.

RTF_REJECT

Instead of forwarding a packet like a normal route, routes with RTF_REJECT cause packets to be dropped and **unreachable** messages to be sent to the packet originators. This flag is only valid on routes pointing at the *loopback* interface.

RTF_BLACKHOLE

Like the RTF_REJECT flag, routes with RTF_BLACKHOLE cause packets to be dropped, but **unreachable** messages are not sent. This flag is only valid on routes pointing at the *loopback* interface.

RTF_STATIC

When GateD starts, it reads all the routes currently in the kernel forwarding table. Besides interface routes, it usually marks everything else as a *remnant* from a previous run of GateD and deletes it after a few minutes. This means that routes added with the *route* command will not be retained after GateD has started.

To fix this, the RTF_STATIC flag was added. When the route command is used to install a route that is not an interface route it sets the RTF_STATIC flag. This signals to GateD that said route was added by the systems administrator and should be retained.

Kernel statement

```
kernel {
    options
        [ nochange ]
        [ noflushatexit ]
        ;
    routes number ;
    flash
        [ limit number ]
        [ type interface | interior | all ]
        ;
    background
        [ limit number ]
        [ priority flash | higher | lower ]
        ;
    traceoptions trace_options ;
} ;
```

options *option_list*

Configure kernel options. The valid options are:

nochange

On systems supporting the routing socket this insures that changes operations will not be performed, only deletes and adds. This is useful on early versions of the routing socket code where the change operation was broken.

noflushatexit

During normal shutdown processing GateD deletes all routes from the kernel forwarding table that do not have a **retain** indication. The **noflushatexit** option prevents route deletions at shutdown. Instead, routes are changed and added to make sure that all the routes marked with **retain** get installed.

This is handy on systems with thousands of routes. Upon start-up, GateD will notice which routes are in the kernel forwarding table and not add them back.

routes *number*

On some systems kernel memory is at a premium. With this parameter a limit can be placed on the maximum number of routes GateD will install in the kernel. Normally GateD adds/changes/deletes routes in interface/internal/external order, i.e. it queues interface routes first, followed by internal routes, followed by external routes, and processes the queue from the beginning. If a this parameter is specified and the limit is hit, GateD does two scans of the list instead. On the first scan it does deletes, and also deletes all changed routes, turning the queued changes into adds. It then rescans the list doing adds in interface/internal/external order until it hits the limit again. This will tend to favor internal routes over external routes. The default is not to limit the number of routes in the kernel forwarding table.

flash

When routes change, the process of notifying the protocols is called a *flash update*. The kernel forwarding table interface is the first to be notified. Normally a maximum of 20 interface routes may be processed during one flash update. The **flash** command allows tuning of these parameters.

limit *number*

Specifies the maximum number of routes which may be processed during one flash update. The default is **20**. A value of **-1** will cause all pending route changes of the specified type to be processed during the flash update.

type interface | interior | all

Specifies the type of routes that will be processed during a flash update. **Interior** specifies that interior routes will also be installed. See the definition of interior gateway protocols (page 2-21). **All** specifies the inclusion of exterior routes as well. See the definition of exterior gateway protocols (page 2-21). The default is **interface** which specifies that only interface routes will be installed during a flash update.

Specifying **flash limit -1 all** causes all routes to be installed during the flash update; this mimics the behavior of previous versions of GateD.

background

Since only interface routes are normally installed during a flash update, the remaining routes are processed in batches in the background, that is, when no routing protocol traffic is being received. Normally, 120 routes are installed at a time to allow other tasks to be performed and this background processing is done at lower priority than flash updates the following parameters allow tuning of these parameters:

limit *number*

Specifies the number of route which may be processed at during one batch. The default is 120.

priority flash | higher | lower

Specifies the priority of the processing of batches of kernel updates in relationship to the *flash update* processing. The default is **lower** which means that flash updates are processed first. To process kernel updates at the same priority as flash updates, specify **flash**; to process them at a lower priority, use **lower**.

Tracing options

While the kernel interface is not technically a routing protocol, in many cases it is handled as one. The following two options make sense when entered from the command line since the code that uses them is executed before the trace file is parsed.

symbols

Symbols read from the kernel, by `nlist`() or similar interface.

iflist

Interface list scan. This option is useful when entered from the command line as the first interface list scan is performed before the configuration file is parsed.

The following tracing options may only be specified in the configuration file. They are not valid from the command line.

remnants

Routes read from the kernel when GateD starts.

request

Requests by GateD to Add/Delete/Change routes in the kernel forwarding table.

The following general option and packet-tracing options only apply on systems that use the routing socket to exchange routing information with the kernel. They do not apply on systems that use the old BSD4.3 `ioctl`() interface to the kernel.

info

Informational messages received from the routing socket, such as TCP loss, routing lookup failure, and route resolution requests. GateD does not currently do processing on these messages, just logs the information if requested.

Packet tracing options (which may be modified with **detail**, **send** and **recv**):

routes

Routes exchanged with the kernel, including Add/Delete/Change messages and Add/Delete/ Change messages received from other processes.

redirect

Redirect messages received from the kernel.

interface

Interface status messages received from the kernel. These are only supported on systems with networking code derived from BSD 4.4.

other

Other messages received from the kernel, including those mentioned in the *info* type above.

Static statements

Static statements define the static routes used by GateD. A single **static** statement can specify any number of routes. The **static** statements occur after protocol statements and before control statements in the `gated.conf` file. Any number of **static** statements may be specified, each containing any number of static route definitions. These routes can be overridden by routes with better preference values.

```
static {
    ( host host ) | default |
    ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
        gateway gateway_list
        [ interface interface_list ]
        [ preference preference ]
        [ retain ]
        [ reject ]
        [ blackhole ]
        [ noinstall ] ;
    ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
        interface interface
        [ preference preference ]
        [ retain ]
        [ reject ]
        [ blackhole ]
        [ noinstall ] ;
} ;
```

host *host* **gateway** *gateway_list*
(*network* [(**mask** *mask*) | ((**masklen** | /) *number*)])
default gateway *gateway_list*

This is the most general form of the static statement. It defines a static route through one or more gateways. Static routes are installed when one or more of the **gateways** listed are available on directly attached interfaces. If more than one eligible gateway is available, these are limited by the number of multipath destinations supported (this compile-time parameter is currently almost always one on Unix).

Parameters for static routes are:

interface *interface_list*

When this parameter is specified, gateways are only considered valid when they are on one of these interfaces. See the section on interface list specification (page 2-14) for the description of the *interface_list*.

preference *preference*

This option selects the preference of this static route. The preference controls how this route competes with routes from other protocols. The default preference is 60.

retain

Normally GateD removes all routes except interface routes from the kernel forwarding table during a graceful shutdown. The **retain** option may be used to prevent specific static routes from being removed. This is useful to insure that some routing is available when GateD is not running.

reject

Instead of forwarding a packet like a normal route, **reject** routes cause packets to be dropped and **unreachable** messages to be sent to the packet originators. Specifying this option causes this route to be installed as a reject route. Not all kernel forwarding engines support reject routes.

blackhole

A **blackhole** route is the same as a **reject** route except that **unreachable** messages are not supported.

noinstall

Normally the route with the lowest preference is installed in the kernel forwarding table and is the route exported to other protocols. When **noinstall** is specified on a route, it will not be installed in the kernel forwarding table when it is active, but it will still be eligible to be exported to other protocols.

(*network* [(**mask** *mask*) | ((**masklen** | **/**) *number*)]) **interface** *interface*

This form defines a static interface route which is used for primitive support of multiple network addresses on one interface. The **preference**, **retain**, **reject**, **blackhole** and **noinstall** options are the same as described above.

Control statements overview

Control statements control routes that are imported from routing peers and routes that are exported to these peers. These are the final statements to be included in the `gated.conf` file.

The control statements are:

- Routing Filtering
- Matching AS Paths
- the Import Statement
- the Export Statement
- the Aggregate Statement
- the Generate Statement

Route filtering

Routes are filtered by specifying configuration language that will match a certain set of routes by destination, or by destination and mask. Among other places, route filters are used on `martians`, `import,` and `export` statements.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

- declaration of a global filter once and reference by name.
- single parse cost, meaning that the structure for the filter will only be created once and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list and reduce the memory requirements if the filter or list is used more than once.

The action taken when no match is found is dependent on the context. For instance, `import` and `export` route filters assume an `all reject ;` at the end of a list.

A route will match the most specific filter that applies. Specifying more than one filter with the same destination, mask, and modifiers will generate an error.

Filtering syntax

network **mask** *mask* [**exact** | **refines** | **between** *number* **and** *number*]
network **masklen** |/ *number* [**exact** | **refines** | **between** *number* **and** *number*]
all
default
host *host*

These are all the possible formats for a route filter. Not all of these formats are available in all places, for instance, the `host` and `default` formats are not valid for `martians`.

In most cases it is possible to specify additional parameters relevant to the context of the filter. For example, on a **martian** statement it is possible to specify the **allow** keyword, on an **import** statement you can specify a preference, and on an **export** statement you can specify a metric.

network **mask** *mask* [**exact** | **refines** | **between** *number* **and** *number*]
network (**masklen** | /) *number* [**exact** | **refines** | **between** *number* **and** *number*]

Matching usually requires both an address and a mask, although the mask is implied in the shorthand forms listed below. These forms vary in how the mask is specified. In the first, the mask is explicitly specified. In the second, the mask is specified by the number of contiguous one bits.

If no additional parameters are specified, any destination that falls in the range given by the network and mask is matched, the mask of the destination is ignored. If a *natural* network is specified, the network, any subnets, and any hosts will be matched.

The three optional modifiers cause the mask of the destination to be considered also:

exact

This parameter specifies that the mask of the destination must match the supplied mask *exactly.* This is used to match a network, but no subnets or hosts of that network.

refines

Specifies that the mask of the destination must be more specified (i.e. longer) than the filter mask. This is used to match subnets and/or hosts of a network, but not the network.

between *number* **and** *number*

Specifies that the mask of the destination must be as or more specific (as long as or longer) than the lower limit (the first *number* parameter) and no more specific (as long as or shorter) than the upper limit (the second *number* parameter). Note that **exact** and **refines** are both special cases of **between**.

More than one filter may be specified with a given network and mask, as long as the filters differ in their modifiers. In the case of two otherwise identical filters with the between modifier, the ranges given by the filters must not overlap.

If two filters differ only in terms of their modifiers, the filters will be matched in the following order: **exact**, **between**, **refines**, no modifier. For example, a filter with **exact** specified will be matched before an eligible filter with **between**.

all

This entry matches anything. It is equivalent to:

```
0.0.0.0 mask 0.0.0.0
```

default

Matches the **default** route. To match, the address must be the default address and the mask must be all zeros. This is equivalent to:

```
0.0.0.0 mask 0.0.0.0 exact
```

host *host*

Matches a specific host route. To match, the address must exactly match the specified *host* and the network mask must be a host mask (i.e. all ones). This is equivalent to:

```
host mask 255.255.255 exact
```

Matching AS paths

An AS path is a list of autonomous systems that routing information has passed through to get to this router, and an indicator of the origin of this information. This information can be used to prefer one path to a destination network over another. The primary method for doing this with GateD is to specify a list of patterns to be applied to AS paths when importing and exporting routes.

Each autonomous system through which a route passes prepends its AS number to the beginning of the AS path.

The origin information details the completeness of AS path information. An origin of **igp** indicates the route was learned from an interior routing protocol and is most likely complete. An origin of **egp** indicates the route was learned from an exterior routing protocol that does not support AS paths (EGP, for example) and the path is most likely not complete. When the path information is definitely not complete, an origin of **incomplete** is used.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

- declaration of a global filter once and reference by name
- single parse cost

  This means that the structure for the filter will only be created once, and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list and reduce the memory requirements if the filter or list is used more than once.

AS path regular expressions are defined in RFC 1164, section 4.2.

AS path matching syntax

An AS path is matched using the following syntax.

aspath *aspath_regexp* **origin any** | ([**igp**] [**egp**] [**incomplete**])

This specifies that an AS matching the *aspath_regexp* with the specified origin is matched.

AS path regular expressions

Technically, an AS path regular expression is a regular expression with the alphabet being the set of AS numbers. An AS path regular expression is composed of one or more AS-path expressions. An AS path expression is composed of AS path terms and AS path operators.

AS path terms

An AS path term is one of the following three objects:

```
autonomous_system
.
( aspath_regexp )
```

autonomous_system

Is any valid autonomous system number, from one through 65534 inclusive.

.

Matches any autonomous system number.

(*aspath_regexp*)

Parentheses group subexpressions -- an operator such as `*` or `?` works on a single element, or on a regular expression enclosed in parentheses

AS path operators

An AS path operator is one of the following:

```
aspath_term {m,n}
aspath_term {m}
aspath_term {m,}
aspath_term *
aspath_term +
aspath_term ?
aspath_term null
aspath_term | aspath_term
```

aspath_term {**m,n**}

A regular expression followed by `{m,n}` (where m and n are both non-negative integers and m <= n) means at least *m* and at most *n* repetitions.

aspath_term {**m**}

A regular expression followed by `{m}` (where m is a positive integer) means exactly *m* repetitions.

aspath_term {**m,**}

A regular expression followed by `{m,}` (where m is a positive integer) means *m or more* repetitions.

aspath_term *****

An AS path term followed by `*` means *zero or more* repetitions. This is shorthand for `{0,}`.

aspath_term +

A regular expression followed by `+` means *one or more* repetitions. This is shorthand for `{1,}`.

aspath_term **?**

A regular expression followed by `?` means *zero or one* repetition. This is shorthand for `{0,1}`.

aspath_term **null**

Matches all things with a null aspath. Cannot be combined with other atoms. Example usage: **proto bgp aspath** (**null**) **origin any** { **all**; };

aspath_term | *aspath_term*

Matches the AS term on the left, or the AS term on the right.

AS path attributes

BGP updates carry a number of *path attributes*. Some of these, like the AS path, are required. Others are optional and may or may not appear in any given BGP update.

The **aspath-opt** option to the group clause, and its variant the **mod-aspath** option, can be used to generate optional path attributes. The **mod-aspath** option is used in exports to set a community value. Currently only the **community** attribute is supported. The **aspath-opt** attribute may also be used on the import clause to allow optional attributes to be considered when determining GateD's preference for the routes in a particular BGP update.

The syntax of **aspath-op**t (and **mod-aspath**) is as follows:

```
aspath-opt {
    [ community autonomous_system : community-id | community-id ]
    [ community no-export | no-advertise | no-export-subconfed | none ]
}
mod-aspath {
    [ community autonomous_system : community-id |community-id ]
    [ community no-export | no-advertise | no-export-subconfed ]
    [ comm-split autonomous_system community-id ]
}
```

Communities

Communities may be specified as an AS and a community ID or as one of the distinguished special communities (with the **community** keyword).

When originating BGP communities, the set of communities which is actually sent is the union of the communities received with the route (if any), those specified in group policy (if any) and those specified in export policy (if any).

When receiving BGP communities, the update is only matched if *all* communities specified in aspath-opt are present in the BGP update. (If additional communities are also present in the update, it will still be matched.)

There is a limit of 25 communities in any single policy clause. This limit may be increased at compile time by increasing the value of AS_COMM_MAX. :

community *autonomous_system* : *community_id*

This causes a community "tag" to be added to the transmitted path attributes. The *autonomous_system* part of the community should be set to the local AS unless there is a specific need to do otherwise. This associates an AS with a community.

community *community_id*

If one only specifies the *community_id*, the *autonomous_system* of the advertising router is implicitly prepended.

Note: For backward compatibility, the [**comm-split** `autonomous_system` *`community-id`*] is interchangeable.

community no-export

This is a special community which indicates that the routes associated with this attribute must not be advertised outside a BGP AS boundary.

community no-advertise

This is a special community which indicates that the routes associated with this attribute must not be advertised to other BGP peers.

community no-export-subconfed

This is a special community which indicates that the routes associated with this attribute must not be advertised to external BGP peers.

community none

This is not actually a community, but rather a keyword which specifies that a received BGP update is only to be matched if no communities are present. It has no effect when originating communities; therefore can only be used with **aspath-opt**.

The import statement

Importation of routes from routing protocols and installation of the routes in GateD's routing database is controlled by **import** statements. The format of an **import** statement varies depending on the source protocol.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

– declaration of a global filter once and reference by name.

– single parse cost

This means that the structure for the filter will only be created once, and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list, and reduce the memory requirements if the filter or list is used more than once.

Specifying precedences and preferences

In all cases, one of three keywords may be specified to control how routes compete with other protocols:

restrict
precedence *precedence*
preference *preference*
localpref *preference*

restrict

Specifies that the routes are not desired in the routing table. In some cases this means that the routes are not installed in the routing table. In others it means that they are installed with a negative preference. This prevents them from becoming active, and they will not be installed in the forwarding table or exported to other protocols.

precedence *precedence*

Specifies the precedence value used when comparing this route to other routes from other protocols. The route with the lowest precedence available at any given route becomes the active route, is installed in the forwarding table, and is eligible to be exported to other protocols. It is ***not*** recommended that precedence ever be used in any other fashion than that given by the default values.

preference *preference*

Specifies the preference value used when comparing this route to other routes from the same protocol. The route with the lowest preference available at any given route becomes the active route, is installed in the forwarding table, and is eligible to be exported to other protocols. The default preferences is 0 (zero).

localpref *preference*

Specifies the local preference value used when comparing this route to other routes known within the **BGP** protocol. The route with the most preferred local preference available at any given route becomes the active route, is installed in the forwarding table, and is eligible to be exported to other protocols. The default preference is 100.

Route filters

All the formats allow route filters as shown below. See the section on route filters (page 2-82) for a detailed explanation of how they work.

When no route filtering is specified (i.e. when **restrict** is specified on the first line of a statement), all routes from the specified source will match that statement. If any filters are specified, only routes that match the specified filters will be imported. Put differently, if any filters are specified, an **all restrict** is assumed at the end of the list.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

– declaration of a global filter once and reference by name.

– single parse cost, meaning that the structure for the filter will only be created once and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list and reduce the memory requirements if the filter or list is used more than once.

```
network mask mask [exact | refines | between number and number ]
network ( masklen | / ) number
                [ exact | refines | between number and number ]
default
host host
```

Importing routes from BGP and EGP

```
import proto bgp | egp autonomoussystem autonomous_system
    [ subgroup integer ]  [ aspath-opt ] restrict ;
import proto bgp | egp autonomoussystem autonomous_system
    [ subgroup integer ] [ aspath-opt ] [ preference preference ] {
    route_filter  [ restrict | ( precedence  precedence )
| ( preference preference ) | ( localpref preference ) ] ;
} ;


import proto bgp aspath  aspath_regexp
    origin any | ( [ igp ] [egp ] [ incomplete ] )
    [ aspath-opt ] restrict ;
import proto bgp aspath  aspath_regexp
    origin any | ( [ igp ] [egp ] [ incomplete ] )
    [ aspath-opt ] [ preference preference ] {
    route_filter [ restrict | ( precedence  precedence )
| ( preference preference ) | ( localpref preference ) ] ;
} ;
```

EGP importation may be controlled by autonomous system and subgroup. BGP also supports controlling propagation by the use of AS path regular expressions, which are documented in the section on “Matching AS paths” (page 2-84). Note that EGP and BGP versions 2 and 3 only support the propagation of natural networks, so the **host** and **default** route filters are meaningless. BGP version 4 supports the propagation of any destination along with a contiguous network mask.

The **aspath-opt** option allows the specification of import policy based on the path attributes found in the BGP update. (The option is not usable with EGP.) If multiple communities are specified in the **aspath-opt** option, only updates carrying all of the specified communities will be matched. If none is specified, only updates lacking the community attribute will be matched.

Note that it is quite possible for several BGP import clauses to match a given update. If more than one clause matches, the first matching clause will be used. All later matching clauses will be ignored. For this reason, it is generally desirable to order import clauses from most to least specific. An import clause without an **aspath-opt** option will match any update with any (or no) communities.

EGP and BGP both store any routes that were rejected implicitly by not being mentioned in a route filter, or explicitly with the **restrict** keyword in the routing table with a negative preference. A negative preference prevents a route from becoming active, which prevents it from being installed in the forwarding table or exported to other protocols. This alleviates the need to break and re-establish a session upon reconfiguration if importation policy is changed.

Importing routes from RIP, HELLO and Redirects

```
import proto rip |  hello  |  redirect
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    restrict ;
import proto rip  |  hello  |  redirect
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    [ preference preference ] {
    route_filter  [ restrict | ( precedence  precedence )
| ( preference preference ) | ( localpref preference ) ] ;
} ;
```

The importation of RIP, HELLO and Redirect routes may be controlled by protocol, source interface and source gateway. If more than one is specified, they are processed from most general (protocol) to most specific (gateway).

RIP and HELLO do not support the use of preference to choose between routes of the same protocol. That is left to the protocol metrics. These protocols do not save routes that were rejected since they have short update intervals.

Importing routes from OSPF

```
import proto ospfase  [ tag ospf_tag ] restrict ;
import proto ospfase  [ tag ospf_tag ]
    [ preference preference ] {
    route_filter  [ restrict | ( precedence  precedence )
| ( preference preference ) | ( localpref preference ) ] ;
} ;
```

Due to the nature of OSPF, only the importation of ASE routes may be controlled. OSPF intra-area and inter-area routes are always imported into the GateD routing table with a preference of 10. If a tag is specified, the import clause will only apply to routes with the specified tag.

It is only possible to restrict the importation of OSPF ASE routes when functioning as an AS border router. This is accomplished by specifying an **export ospfase** clause. Specification of an empty export clause may be used to restrict importation of ASEs when no ASEs are being exported.

Like the other interior protocols, preference can not be used to choose between OSPF ASE routes, that is done by the OSPF costs. Routes that are rejected by policy are stored in the table with a negative preference.

The export statement

The **import** statement controls which routes received from other systems are used by GateD, and the **export** statement controls which routes are advertised by GateD to other systems. Like the **import** statement, the syntax of the **export** statement varies slightly per protocol. The syntax of the **export** statement is similar to the syntax of the **import** statement, and the meanings of many of the parameters are identical. The main difference between the two is that while route importation is just controlled by source information, route exportation is controlled by both destination and source.

The outer portion of a given **export** statement specifies the destination of the routing information you are controlling. The middle portion restricts the sources of importation that you wish to consider. And the innermost portion is a route filter used to select individual routes.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

- declaration of a global filter once and reference by name.
- single parse cost

This means that the structure for the filter will only be created once, and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list, and reduce the memory requirements if the filter or list is used more than once.

Specifying metrics

The most precise specification of a metric is the one applied to the route being exported. The values that may be specified for a metric depend on the destination protocol that is referenced by this **export** statement.

```
restrict
[all] metric [igp] metric
localpref preference
```

restrict

Specifies that nothing should be exported. If specified on the destination portion of the **export** statement, it specifies that nothing at all should be exported to this destination. If specified on the source portion, it specifies that nothing from this source should be exported to this destination. If specified as part of a route filter, it specifies that the routes matching that filter should not be exported.

metric *metric*

Specifies the metric to be used when exporting to the specified destination.

all metric igp is used for exporting (redistributing) the IGP metric as a BGP MED.

Here is an example that exports the OSPF metric as the BGP MED:

```
export proto bgp as foo {
                proto ospf {
                        all metric igp;
                };
        };
```

Note: If this is used with BGP, a BGP update is issued *immediately* upon the IGP metric changing.

This means that a flapping link within a network could cause substantial BGP route flap, with attendant damping being likely upstream. Non-pathological cases of this may be covered by also using the **outdelay** variable in the **BGP** statement.

localpref *preference*

This is used for setting the local preference to be export to other IBGP peers. It can be set anywhere metric can be set in the **export** statement.

Here are examples, if my AS is *bar*; thus IBGP, this exports the OSPF routes into BGP with the **localpref** of 150:

```
export proto bgp as bar {
                proto ospf  localpref 150 {
                        all ;
                };
        };
```

This example exports the RIP routes into BGP with the **localpref** of 170.

```
export proto bgp as bar {
                proto rip {
                        all localpref 170;
                };
        };
```

Route filters

All the formats allow route filters as shown below. See the section (page 2-82) on Route Filtering for a detailed explanation of how they work. When no route filtering is specified (i.e. when **restrict** is specified on the first line of a statement), all routes from the specified source will match that statement. If any filters are specified, only routes that match the specified filters will be exported. Put differently, if any filters are specified, an `all restrict ;` is assumed at the end of the list.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

– declaration of a global filter once and reference by name

– single parse cost

This means that the structure for the filter will only be created once, and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list, and reduce the memory requirements if the filter or list is used more than once.

network **mask** *mask* [**exact** | **refines** | **between** *number* **and** *number*]

network (**masklen** | /) *number*
[**exact** | **refines** | **between** *number* **and** *number*]

default

host *host*

Specifying the destination

As mentioned above, the syntax of the **export** statement varies depending on the protocol it is being applied to. A requirement that applies in all cases is the specification of a metric. All protocols define a default metric to be used for routes being exported. In most cases this can be overridden at several levels of the **export** statement.

The specification of the source of the routing information being exported (the *export_list*) is described below.

Exporting to EGP and BGP

```
export proto bgp | egp as autonomous system
    restrict ;
export proto bgp | egp as autonomous system [ mod-aspath]
    [ metric metric  | localpref preference]  [ subgroup integer] {
    export_list ;
} ;
```

Exportation to EGP and BGP is controlled by autonomous system (AS), the same policy is applied to all routers in the AS or subgroup. EGP metrics range from 0 to 255, inclusive, with 0 being the most attractive.

BGP metrics are 16-bit unsigned quantities, i.e. they range from 0 to 65535, inclusive, with 0 being the most attractive. While BGP version 4 actually supports 32-bit unsigned quantities, GateD does not yet support this. In BGP version 4, the metric is otherwise known as the Multi-Exit Discriminator, or MED.

In BGP, the **mod-aspath** option may be used to send the BGP community attribute. Any communities specified with the **mod-aspath** option are sent in addition to any received with the route or specified in the group statement.

If no export policy is specified, only routes to attached interfaces will be exported. If any policy is specified, the defaults are overridden; it is necessary to explicitly specify everything that should be exported.

Note that EGP and BGP versions 2 and 3 only support the propagation of natural networks, so the **host** and **default** route filters are meaningless. BGP version 4 supports the propagation of any destination along with a contiguous network mask.

Exporting to RIP and HELLO

```
export proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    restrict ;
export proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    [ metric metric ] {
    export_list ;
} ;
```

Exportation to RIP and HELLO is controlled by protocol, interface, or gateway. If more than one is specified, they are processed from most general (protocol) to most specific (gateway).

It is not possible to set metrics for exporting RIP routes into RIP, or exporting HELLO routes into HELLO. Attempts to do this are silently ignored.

If no export policy is specified, RIP and interface routes are exported into RIP and HELLO, and interface routes are exported into HELLO. If any policy is specified, the defaults are overridden. It is necessary to explicitly specify everything that should be exported.

RIP version 1 and HELLO assume that all subnets of the shared network have the same subnet mask so they are only able to propagate subnets of that network. RIP version 2 removes that restriction and is capable of propagating all routes when not sending version 1-compatible updates.

Note: To announce routes from other protocols (i.e., IP aliases to the loopback interface, static routes, internally generated default routes, etc.) that do not have a **metric** associated with them, via RIP or HELLO, it is necessary to specify the **metric** at some level in the **export** clause. Just setting a default metric for RIP or HELLO is not sufficient. This is a safeguard to verify that the announcement is intended.

Exporting to OSPF

```
export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
    restrict ;
export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
    [ metric metric ] {
    export_list ;
} ;
```

Note: Keep the the order of the OSPF export statements as shown, reversing them causes a parser error.

It is not possible to create OSPF intra- or inter-area routes by exporting routes from the GateD routing table into OSPF. It is only possible to export from the GateD routing table into OSPF

ASE routes. It is also not possible to control the propagation of OSPF routes within the OSPF protocol.

There are two types of OSPF ASE routes, **type 1** and **type 2**. See the OSPF protocol configuration for a detailed explanation of the two types (page 2-29). The default type is specified by the **defaults** subclause of the **ospf** clause. This may be overridden by a specification on the **export** statement.

OSPF ASE routes also have the provision to carry a **tag**. This is an arbitrary 32-bit number that can be used on OSPF routers to filter routing information. See the OSPF protocol configuration for detailed information on OSPF tags (page 2-29). The default tag specified by the ospf defaults clause may be overridden by a tag specified on the **export** statement.

Specifying the source

The export list specifies export based on the origin of a route, the syntax varies depending on the source.

Exporting BGP and EGP routes

```
proto bgp | egp autonomoussystem autonomous_system  [subgroup integer ]
    restrict ;
proto bgp | egp autonomoussystem autonomous_system [subgroup integer ]
    [ metric metric  | localpref preference ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

BGP and EGP routes may be specified by source autonomous system. All routes may be exported by **aspath**, see below for more information.

Exporting RIP and HELLO routes

```
proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    restrict ;
proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

RIP and HELLO routes may be exported by protocol, source interface and/or source gateway.

Exporting OSPF routes

```
proto ospf | ospfase restrict ;
proto ospf | ospfase [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Both OSPF and OSPF ASE routes may be exported into other protocols. See below for information on exporting by **tag**.

Exporting routes from non-routing protocols

Non-routing with interface

```
proto direct | static | kernel
    [ (interface interface_list ) ]
    restrict ;
proto direct | static | kernel
    [ (interface interface_list ) ]
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

These protocols may be exported by protocol, or by the interface of the next hop. These protocols are:

direct

Routes to directly-attached interfaces.

static

Static routes specified in a **static** clause.

kernel

On systems with the routing socket, routes learned from the routing socket are installed in the GateD routing table with a protocol of **kernel**. These routes may be exported by referencing this protocol. This is useful when it is desirable to have a script install routes with the **route** command and propagate them to other routing protocols.

Non-routing by protocol

```
proto default | aggregate
    restrict ;
proto default | aggregate
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

These protocols may only be referenced by protocol.

default

Refers to routes created by the **gendefault** option. It is recommended that route generation be used instead.

aggregate

Refers to routes synthesized from other routes when the aggregate and generate statements are used. See the section on Route Aggregation (page 2-100) for more information.

Exporting by AS path

```
proto proto | all aspath aspath_regexp
    origin any | ( [ igp ] [egp ] [ incomplete ] )
    restrict ;
proto proto | all aspath aspath_regexp
    origin any | ( [ igp ] [egp ] [ incomplete ] )
    [ metric metric  | localpref preference ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

When BGP is configured, all routes are assigned an AS path when they are added to the routing table. For all interior routes, this AS path specifies IGP as the origin and no ASes in the AS path (the current AS is added when the route is exported). For EGP routes, this AS path specifies EGP as the origin and the source AS as the AS path. For BGP routes, the AS path is stored as learned from BGP.

AS path regular expressions are documented in the section on *Matching AS Paths* (page 2-84).

Exporting by route Tag

```
proto proto | all tag tag restrict ;
proto proto | all tag tag
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Both OSPF and RIP version 2 currently support **tags**, all other protocols always have a tag of zero. The source of exported routes may be selected based on this tag. This is useful when routes are classified by tag when they are exported into a given routing protocol.

Route aggregation and generation statements

Route aggregation is a method of generating a more general route given the presence of a specific route. It is used, for example, at an autonomous system border to generate a route to a network to be advertised via EGP given the presence of one or more subnets of that network learned via RIP.

Older versions of GateD automatically performed this function, generating an aggregate route to a natural network (using the old Class A, B and C concept) given an interface to a subnet of that natural network. However, that was not always the correct thing to do, and with the advent of classless inter-domain routing it is even more frequently the wrong thing to do, so aggregation must be explicitly configured. No aggregation is performed unless explicitly requested in an **aggregate** statement.

Route aggregation is also used by regional and national networks to reduce the amount of routing information passed around. With careful allocation of network addresses to clients, regional networks can just announce one route to regional networks instead of hundreds.

Aggregate routes are not actually used for packet forwarding by the originator of the aggregate route, only by the receiver (if it wishes). A router receiving a packet which does not match one of the component routes which led to the generation of an aggregate route is supposed to respond with an ICMP network unreachable message. This is to prevent packets for unknown component routes from following a default route into another network where they would be forwarded back to the border router, and around and around again and again, until their TTL expires. Sending an unreachable message for a missing piece of an aggregate is only possible on systems with support for reject routes.

A slight variation of aggregation is the generation of a route based on the existence of certain conditions. This is sometimes known as the *route of last resort*. This route inherits the next hops and aspath from the contributor specified with the lowest (most favorable) preference. The most common usage for this is to generate a default based on the presence of a route from a peer on a neighboring backbone.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

– declaration of a global filter once and reference by name.

– single parse cost

This means that the structure for the filter will only be created once and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list and reduce the memory requirements if the filter or list is used more than once.

Aggregation and generation syntax

```
aggregate default
    | ( network [ ( mask mask ) | ( ( masklen number |/ ) number ) ] )
    [ preference preference ] [ brief | truncate ] {
    proto [ all | direct | static | kernel | aggregate | proto ]
        [ ( as autonomous_system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        restrict ;
    proto [ all | direct | static | kernel | aggregate | proto ]
        [ ( as autonomous_system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        [ preference preference ] {
        route_filter [ restrict | ( preference preference ) ] ;
    } ;
} ;
generate default
    | ( network [ ( mask mask ) | ( ( masklen number | / ) number ) ) ] )
    [ preference preference ] {
        [ ( as autonomous_system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        restrict ;
    proto [ all | direct | static | kernel | aggregate | proto ]
        [ ( as autonomous_system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        [ preference preference ] {
        route_filter [ restrict | ( preference preference ) ] ;
    } ;
} ;
```

Routes that match the route filters are called *contributing* routes. They are ordered according to the aggregation preference that applies to them. If there is more than one contributing route with the same aggregating preference, the route's own preferences are used to order the routes. The preference of the aggregate route will be that of the contributing route with the lowest aggregate preference.

preference *preference*

Specifies the preference to assign to the resulting aggregate route. The default preference is 130.

brief

Used to specify that the AS path should be truncated to the longest common AS path. The default is to build an AS path consisting of SETs and SEQUENCEs of all contributing AS paths. The ATOMIC_AGGREGATE path attribute is set on truncated AS paths.

truncate

Used to specify that the AS path should be completely truncated, removing all elements of contributing AS paths. The truncated AS path is used even if there is only one contributing AS path. The ATOMIC_AGGREGATE path attribute is set on truncated AS paths.

proto *proto*

In addition to the special protocols listed, the contributing protocol may be chosen from among any of the ones supported by (and currently configured into) GateD.

as *autonomous_system*

Restrict selection of routes to those learned from the specified autonomous system.

tag *tag*

Restrict selection of routes to those with the specified tag.

aspath *aspath_regexp*

Restrict selection of routes to those that match the specified AS path.

restrict

Indicates that these routes are not to be considered as contributors of the specified aggregate. The specified protocol may be any of the protocols supported by GateD.

route_filter

See below.

A route may only contribute to an aggregate route which is more general than itself. It must match the aggregate under its mask. Any given route may only contribute to one aggregate route, which will be the most specific configured, but an aggregate route may contribute to a more general aggregate.

Route filters

All the formats allow route filters as shown below. See the section on Route Filters (page 2-82) for a detailed explanation of how they work.

When no route filtering is specified (i.e. when **restrict** is specified on the first line of a statement), all routes from the specified source will match that statement. If any filters are specified, only routes that match the specified filters will be considered as contributors. Put differently, if any filters are specified, an `all restrict ;` is assumed at the end of the list.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **define** filter statement. It gives the advantage of:

– declaration of a global filter once and reference by name.

– single parse cost

This means that the structure for the filter will only be created once, and the use of a pointer to the structure will reduce the CPU cycles to parse the filter or list and reduce the memory requirements if the filter or list is used more than once.

```
network mask mask [exact | refines | between number and number ]
network (masklen | /) number [ exact | refines | between number
        and number ]
default
host host
```

GateD State Monitor (GSM)

The GateD State Monitor (GSM) provides an interactive interface to a running GateD daemon which can be used to query internal **gated** variables. This interface, which is implemented like any other protocol in Gated, accepts telnet connections to port 616 (C.f. [RFC 854] for a description of the telnet protocol), and, after user identification, answers any query sent as ASCII commands. The commands include querying about the memory, routing table, interface list and other internal parameters.

Password

GSM will identify the user by using the UNIX password of the *netstar* ID of the system. In other words, to use the GSM interface, one must have a '`netstar`' account and set up its password that will be used when connecting to the GSM.

GSM connection options

There are two ways to establish a GSM interface, through a direct telnet connection or using the CLI **gsm** command, which establishes a telnet connection for you.

CLI interface

When you enter GSM from the CLI prompt, **gsm** returns information about the GateD running on that local machine. To obtain GateD statistics for a different GRF system, you must access that GateD through the UNIX shell, shell usage is described below.

You need the password from the administrative 'netstar' account, the default password shipped with your system is "`NetStar`". If you changed the administrative password, use that new one.

```
prompt> gsm
Trying 198.xx.xx.1...
Connected to 198.xx.xx.1.
Escape character is '^]'.
Password?-------- ( for example, NetStar )
```

You will see the GSM interface starting, notice that the **gsm** prompt uses the machine domain name:

```
1GRF Gated State Monitor. Version GateD R3_5Beta_3; CVS Branch:A1_4_1;
Path:/
/gated/code/GSM
GateD-router.sitename.com>
```

Refer to the Commands chapter in this manual for examples of **gsm** output.

telnet connection

Open a telnet connection to the machine running GateD on TCP port 616. You can telnet from the administrative LAN or from the GateD machine itself. To telnet to the GSM interface, one must use the password from the administrative 'netstar' account. The default password = "NetStar". If you have changed the password, use the new one.

If GateD is running on 192.22.22.22, here is the command:

```
telnet -a  192.22.22.22 616
Trying 192.22.22.22...
Connected to 192.22.22.22.
Escape character is '^]'.
Password? -------   ( for example, NetStar )
1GRF Gated State Monitor. Version GateD R3_5Beta_3;
Path:
GateD-mike.netstar.com>
```

Top-level commands

When you have the GSM prompt, use **help** to list the top-level commands. Note that the GSM prompt includes the machine's domain name. The top level command **help** or **?** provides a list of available commands:

```
GateD-router.sitename.com> help
1GRF HELP: The possible commands are:
1GRF    ?     : Print help messages
1GRF    help  : Print help messages
1GRF    show  : Show internal values
1GRF    quit  : Close the session
1GRF    enable: Enable the session
1GRF    exec  : Execute actions (must be enabled)
1GRF    exit  : Close the session
```

Abbreviating commands

An abbreviation is accepted provided it unambiguously identifies an entity. Look at the IP summary:

```
GateD-router.sitename.com> sh ip sum
1GRF IP radix tree: 38 nodes, 20 routes
```

Capitalization indicates shortest form of command that will work:

Example:

```
GateD-router.sitename.com> s ip al
```

NOTE: The value x.x.x/len can be replaced with "0" to show all:

Example:

```
GateD-router.sitename.com> s ip ref b 0    # show all routes
                                          # learned via BGP
```

Using the help command

The top level command **help | ?** provides a list of available commands. Commands may be followed by subcommands.

To get help about the subcommand, just type the main command. GSM will display a list of available subcommands. Commands may be abbreviated when no confusion is possible (commands are uniquely identified).

Some examples:

```
GateD-foo.bar.net> show
1GRF HELP: The possible subcommands are:
1GRF    version  : Show the current GateD version
1GRF    kernel   : Show the Kernel support
1GRF    interface [name|index]: Show interface status
1GRF    memory   : Show the memory allocation
1GRF    ip       : Show info about IP protocol
1GRF    task     : Show list of active tasks
1GRF    ospf     : Show info about OSPF protocol
1GRF    timer    : Show list of timers
1GRF    bgp      : Show info about BGP protocol
1GRF    rip      : Show info about rip protocol
1GRF    isis     : Show info about ISIS protocol
GateD-foo.bar.net>

GateD-foo.bar.net> show ip
1GRF    sum       Show info about IP routes
1GRF    exact     [x.x.x/len]: Show info about specific IP routes
1GRF    all       Show entire FIB
1GRF    consume   [x.x.x/len]: Show consuming route and more specic route
1GRF    lessspec   Show less specific routes per protocol
1GRF    refines    Show more specific routes per protocol
GateD-foo.bar.net>
```

The following is applicable for the **lessspec** or **refines** sub-commands:

```
GateD-foo.bar.net>show ip refines
1GRF HELP: The possible subcommands are:
1GRF    ALL    [x.x.x/len]: Show more specific routes from ALL protocols
1GRF    RIP    [x.x.x/len]: Show more specific routes from the RIP protocol
1GRF    STATIC [x.x.x/len]: Show more specific routes from the STATIC proto-
col
1GRF    OSPF   [x.x.x/len]: Show more specific routes from the OSPF protocol
1GRF    ISIS   [x.x.x/len]: Show more specific routes from the ISIS protocol
1GRF    BGP    [x.x.x/len]: Show more specific routes from the BGP protocol
GateD-foo.bar.net>

GateD-foo.bar.net> show ospf
1GRF HELP: The possible subcommands are:
1GRF    summary  : Show OSPF Summary
```

```
1GRF    errors   : Show OSPF Error Counters
1GRF    interface: Show interface status
1GRF    io stats : Show I/O stats
1GRF    nexthops : Show OSPF nexthops
1GRF    ase      : Show OSPF ASE Database
1GRF    lsdb     : Show OSPF LSDB Database
1GRF    border   : Show OSPF ASB/AB RTR Database
GateD-foo.bar.net>

GateD-foo.bar.net> show bgp
1GRF HELP: The possible subcommands are:
1GRF    summary: Show BGP summary
1GRF    peeras [AS number]: Show BGP peer info
1GRF    group  [AS number]: Show BGP group summary
1GRF    aspath : Show BGP aspath info
GateD-foo.bar.net>

GateD-foo.bar.net> show isis
1GRF HELP: The possible subcommands are:
1GRF    summary: Show IS-IS summary
GateD-foo.bar.net>
```

Top-level commands

With the connection established, use **help** to look at the top-level command options:

```
GateD-router.sitename.com> help
1GRF HELP: The possible commands are:
1GRF    ?     : Print help messages
1GRF    help  : Print help messages
1GRF    show  : Show internal values
1GRF    quit  : Close the session
1GRF    enable: Enable the session
1GRF    exec  : Execute actions (must be enabled)
1GRF    exit  : Close the session
```

Second-level commands

Use the **show** command to look at the display options:

```
GateD-router.sitename.com> show
1GRF HELP: The possible show subcommands are:
1GRF    version  : Show the current GateD version
1GRF    kernel   : Show the Kernel support
1GRF    iso      : Show the ISO support
1GRF    interface [name|index]: Show interface status
1GRF    memory   : Show the memory allocation
1GRF    ip       : Show info about IP protocol
1GRF    task     : Show list of active tasks
1GRF    ospf     : Show info about OSPF protocol
1GRF    timer    : Show list of timers
```

```
1GRF    bgp      : Show info about BGP protocol
1GRF    rip      : Show info about rip protocol
1GRF    isis     : Show info about ISIS protocol
```

Protocol-specific information

Now look at information options for a specific protocol, in this case, IP:

```
GateD-router.sitename.com> show ip
1GRF HELP: The possible subcommands are:
1GRF    sum       Show info about IP routes
1GRF    exact     [x.x.x/len]: Show info about specific IP routes
1GRF    aggregate [x.x.x/len]: Show info about specific Aggregate routes
1GRF    all       Show entire FIB
1GRF    consume   [x.x.x/len]: Show consuming route and more specic route
1GRF    lessspec   Show less specific routes per protocol
1GRF    refines    Show more specific routes per protocol
```

Viewing the route table

The recommended way to view IP route tables is through GSM. Establish a GSM session and use the `show ip all` command:

Display the IP route table:

```
GateD-router.sitename.com> sh ip all
1GRF Sta        78.78.78/24 203.3.1.153     IGP (Id 1)
1GRF Sta        99.99.98/24 202.1.1.153     IGP (Id 1)
1GRF Sta        99.99.99/24 212.1.3.152     IGP (Id 1)
1GRF Sta             127/8  127.0.0.1       IGP (Id 1)
1GRF Sta      198.174.11/24 206.146.160.1   IGP (Id 1)
1GRF Dir         202.1.1/24 202.1.1.151     IGP (Id 1)
1GRF Dir         202.1.2/24 202.1.2.151     IGP (Id 1)
1GRF Dir         202.1.3/24 202.1.3.151     IGP (Id 1)
1GRF Dir         202.5.2/24 202.5.2.151     IGP (Id 1)
1GRF Dir         202.5.4/24 202.5.4.151     IGP (Id 1)
1GRF Dir         203.3.1/24 203.3.1.151     IGP (Id 1)
1GRF Dir        204.10.1/24 204.10.1.151    IGP (Id 1)
1GRF Sta   204.100.1.147/32 208.1.1.152     IGP (Id 1)
1GRF Sta     205.2.4.138/32 212.1.2.134     IGP (Id 1)
1GRF Dir     206.146.160/24 206.146.160.151 IGP (Id 1)
1GRF Dir         208.1.1/24 208.1.1.151     IGP (Id 1)
1GRF Dir         212.1.1/24 212.1.1.151     IGP (Id 1)
1GRF Dir         212.1.2/24 212.1.2.151     IGP (Id 1)
1GRF Dir         212.1.3/24 212.1.3.151     IGP (Id 1)
```

Now look at a specific set of routes:

```
GateD-router.sitename.com> sh ip cons 202/8
1GRF Dir         202.1.1/24 202.1.1.151     IGP (Id 1)
1GRF Dir         202.1.2/24 202.1.2.151     IGP (Id 1)
1GRF Dir         202.1.3/24 202.1.3.151     IGP (Id 1)
1GRF Dir         202.5.2/24 202.5.2.151     IGP (Id 1)
1GRF Dir         202.5.4/24 202.5.4.151     IGP (Id 1)
GateD-router.sitename.com> quit
```

Analysis of the Show IP command

This section breaks down the components and options of the **show ip** command:

```
Show IP
  ALl                              # Entire FIB
  Sum                              # Info about IP routes
  Exact    [ipaddr/len | ipaddr]   # Info about specific IP routes
  AGgregate [ipaddr/len | ipaddr]  # Info about specific Aggregate
                                   # routes

  Consume [ipaddr/len | ipaddr]    # More and less specific routes

  Lessspec                     # Less specific routes per protocol
      ALl    [ipaddr/len | ipaddr]
      Rip    [ipaddr/len | ipaddr]
      Static [ipaddr/len | ipaddr]
      Ospf   [ipaddr/len | ipaddr]
      Isis   [ipaddr/len | ipaddr]
      Bgp    [ipaddr/len | ipaddr]
      Aggreg [ipaddr/len | ipaddr]

 REfines                       # More specific routes per protocol
      ALl    [ipaddr/len | ipaddr]
      Rip    [ipaddr/len | ipaddr]
      Rtatic [ipaddr/len | ipaddr]
      Ospf   [ipaddr/len | ipaddr]
      Isis   [ipaddr/len | ipaddr]
      Bgp    [ipaddr/len | ipaddr]
      Aggreg [ipaddr/len | ipaddr]

Hidden x.x.x/len       # All routes in hidden state
Cidr x.x.x/len         # All routes that have no "natural" netmask
```

Frequently-used commands

The following commands have proven to be useful in managing and debugging GateD configurations:

```
s ip al                    # all routes to be installed in kernel
s ip ref <protocol> 0      # all routes learned via <protocol>
s ip ref al xx             # all routes starting with "xx"
s ip exact x.x.x/len       # everything we know about a route
```

Example: show all routes with "99" in first octet

```
GateD-poplar.netstar.com> s ip re al 99
OSP         99.82.1/24 10.2.1.82       (666) IGP (Id 3)
OSP     99.82.82.82/32 10.2.1.82       (666) IGP (Id 3)
Sta        99.175.1/24 206.146.160.1   IGP (Id 2)
Sta  99.175.175.175/32 206.146.160.1   IGP (Id 2)
```

Example: show exact (host route)

```
GateD-poplar.netstar.com> s ip ex  99.82.82.82/32
Route 99.82.82.82 Mask 255.255.255.255 Entries 1 Announced 1 Depth 0 <>
                      Instability Histories:
Proto  Route    NextHop     Proto-prec/Pref Metric/2   Tag Installed
* OSPF_ASE 99.82.82.82  10.2.1.82   150/-   1/30   C0000000
67:49:48
       Forwarding Interface: 10.2.1.175(gf010)
       Status <Int Ext Active Gateway>
       ASPATH (666) IGP (Id 3)
```

Glossary of terms

Terms used in descriptions throughout this document are defined here:

adjacency

A relationship formed between selected neighboring routers for the purpose of exchanging routing information. Not every pair of neighboring routers becomes adjacent.

autonomous system

A set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other ASs.

Since this classic definition was developed, it has become common for a single AS to use several interior gateway protocols and sometimes several sets of metrics within an AS. The use of the term Autonomous System stresses that even when multiple IGPs and metrics are used, the administration of an AS appears to other ASs to have a single coherent interior routing plan and to present a consistent picture of what networks are reachable through it. The AS is represented by a number between 1 and 65534, assigned by the Internet Assigned Numbers Authority.

BGP
Border Gateway Protocol

One of a class of exterior gateway protocols, described in more detail in the BGP section of the Protocol Overview.

cost

An OSPF metric. See **metric**.

delay

A HELLO metric. Valid values are from zero to 30000 inclusive. The value of 30000 is the maximum metric and means unreachable. See **metric**.

designated router

OSPF: Each multi-access network that has at least two attached routers has a designated router. The designated router generates a link state advertisement for the multi-access network and assists in running the protocol. The designated router is elected by the HELLO protocol.

destination

Any network or any host.

distance

An EGP metric. See **metric**. Valid values are from zero to 255 inclusive.

egp
exterior gateway protocol
exterior routing protocol

A class of routing protocols used to exchange routing information within an autonomous system. A detailed explanation of exterior gateway protocols is available in the Protocol Overview.

EGP
Exterior Gateway Protocol

One of a class of exterior gateway protocols, described in more detail in the EGP section of the Protocol Overview.

`FIB`

The table in the kernel that controls the forwarding of packets is a forwarding table, ISO-defined as the forwarding information base.

gateway

An intermediate destination by which packets are delivered to their ultimate destination. A host address of another router that is directly reachable via an attached network. As with any host address it may be specified symbolically.

gateway_list

A list of one or more *`gateways`* separated by white space.

HELLO

One of a class of interior gateway protocols, described in more detail in the HELLO section of the Protocol Overview.

host

The IP address of any host. Usually specified as a dotted quad, four values in the range of 0 to 255 inclusive separated by dots (.). For example `132.236.199.63 or 10.0.0.51.`

It may also be specified as an eight digit hexadecimal string preceded by `0x`. For example `0x????????` or `0x0a000043`. Finally, if `options noresolv` is not specified, a symbolic hostname such as `gated.cornell.edu` or `nic.ddn.mil` is usable. The numeric forms are much preferred over the symbolic form.

interface

The host address of an attached interface. This is the address of a *broadcast*, *nbma* or *loopback* interface and the remote address of a *point-to-point* interface. As with any host address, it may be specified symbolically.

interface

The connection between a router and one of its attached networks.

A physical interface may be specified by a single IP address, domain name, or interface name. (Unless the network is an unnumbered point-to-point network.) Multiple levels of reference in the configuration language allow identification of interfaces using wildcard, interface type name, or delete word address. Be careful with the use of interface names as future Unix operating systems may allow more than one address per interface. Dynamic interfaces can be added or deleted, and indicated as up or down as well as changes to address, netmask and metric parameters.

igp
interior gateway protocol
interior routing protocol

One of a class of routing protocols used to exchange routing information within an autonomous system. A detailed explanation of interior gateway protocols is available in the Protocol Overview.

interface_list

A list of one or more interface names including wildcard names (names without a number) and names which may specify more than one interface or address, or the token "all" for all interfaces. See the section on interface lists for more information (page 2-14).

IS-IS

One of a class of interior gateway protocols, described in more detail in the IS-IS section of the Protocol Overview.

local_address

The host address of an attached interface. This is the address of a *broadcast*, *nbma* or *loopback* interface and the local address of a *point-to-point* interface. As with any host address, it may be specified symbolically.

mask

A means of subdividing networks using address modification. A mask is a dotted quad specifying which bits of the destination are significant. Except when used in a route filter, GateD only supports contiguous masks.

mask length

The number of significant bits in the mask.

metric

One of the units used to help a system determine the best route. Metrics may be based on hop count, routing delay, or an arbitrary value set by the administrator depending on the type of routing protocol. Routing metrics may influence the value of assigned internal preferences. (See **preference**.)

This sample table shows the range of possible values for each routing protocol metric and the value used by each protocol (See Protocol Overview, page 2-21) to reach a destination.

SAMPLE ROUTING PROTOCOL METRICS

Protocol	Metric Represents	Range	Unreachable
RIP	distance (hop-count)	0-15	16
HELLO	delay (milliseconds)	0-29999	30000
OSPF	cost of path	0-?????	Delete
ISIS	cost of path	0-254	Delete
EGP	distance (unused)	0-65535	255
BGP	unspecified	0-65534	65535

multiaccess networks

Those physical networks that support the attachment of multiple (more than two) routers. Each pair of routers on such a network is assumed to be able to communicate directly.

natural mask

neighbor

Another router which with implicit or explicit communication is established by a routing protocol. Neighbors are usually on a shared network, but not always. This term is mostly used in OSPF and EGP. Usually synonymous with *peer*.

neighboring routers

Two routers that have interfaces to a common network. On multi-access networks, routers are dynamically discovered by the OSPF HELLO protocol.

network

Any packet-switched network. A network may be specified by its IP address or network name. The host bits in a network specification must be zero. *Default* may be used to specify the default network (0.0.0.0).

network

The IP address of a network. Usually specified as a dotted quad, one to four values, in the range 0 to 255 inclusive, separated by dots (.), for example, `132.236.199, 132.236` or `10`. It may also be specified as a hexadecimal string preceded by `0x` with an even number of digits, of length between two and eight. For example `0xnnnnnn, 0xnnnn,` or `0x0n`.

Also allowed is the symbolic value `default` which has the distinguished value `0.0.0.0`, the default network. If `options noresolv` is not specified, a symbolic network name such as `nr-tech-prod`, `cornellu-net` and `arpanet` is usable. The numeric forms are preferred over the symbolic form.

number

A positive integer.

OSPF
Open Shortest Path First

One of a class of interior gateway protocols, described in more detail in the OSPF section of the Protocol Overview.

ospf_area

OSPF allows collections of contiguous networks and hosts to be grouped together. Such a group, together with the routes having interfaces to any of the included networks, is called an area.

Each area runs a separate copy of the basic link-state algorithm. This means that each area has its own topological database. The topology of an area is invisible from the outside of the area. Conversely, routers internal to a given area know nothing of the detailed topology external to the area. This isolation of knowledge enables OSPF to effect a marked reduction in routing traffic as compared to treating the entire autonomous system as a single link-state domain.

peer

Another router which with implicit or explicit communication is established by a routingprotocol. Peers are usually on a shared network, but not always. This term is mostly usedby BGP. Usually synonymous with *neighbor.*

port

A UDP or TCP port number. Valid values are from 1 through 65535 inclusive.

precedence (*protocol-precedence*)

A precedence is a value between 0 (zero) and 255, used to set the predecence of routing protocols. The protcol with the best (numerically lowest) precedence contains the prefixes found in the kernel forwarding table and exported to other protocols. A default precedence is assigned to each source from which GateD receives routes (See **preference**.)

preference

A preference is a value between 0 (zero) and 65536, used to select between many routes to the same destination. The route with the best (numerically lowest) preference is the active route. The active route is the one installed in the kernel forwarding table and exported to other protocols. A default preference is assigned to each source from which GateD receives routes; it has the value of 0. (See **precedence**.)

prefix

A contiguous mask covering the most significant bits of an address. The prefix length specifies how many bits are covered.

QoS
quality of service

The OSI equivalent of TOS.

`RIB`

The table that GateD uses internally to store routing information learned from routing protocols is a routing table, ISO-defined as the routing information base.

RIP
Routing Information Protocol

One of a class of interior gateway protocols, described in more detail in the RIP section of the Protocol Overview.

reject route

XXX - define.

route filter

A filter is a configurable entity based on destination address or destination address and mask specified to match a route that is to be filtered out and then ignored when such a route is advertised.

router id

A 32-bit number assigned to each router running the OSPF protocol. This number uniquely identifies the router within the autonomous system.

router_id

An IP address used as unique identifier assigned to represent a specific router.

This is usually the address of an attached interface.

RIB
routing information base
routing database
routing table

The repository of all of GateD's retained routing information, used to make decisions and as a source for routing information that is to be propagated.

simplex

An interface may be marked as simplex either by the kernel, or by interface configuration. A simplex interface is an interface on a broadcast media that is not capable of receiving packets it broadcasts.

GateD takes advantage of interfaces that are capable of receiving their own broadcast packets to monitor whether an interface appears to be functioning properly.

time

A time value, usually a time interval. It may be specified in any one of the following forms:

number

A non-negative decimal number of seconds. For example, `27, 60` or `3600`.

number:number

A non-negative decimal number of minutes followed by a seconds value in the range of zero to 59 inclusive. For example, `0:27`, `1:00` or `60:00`.

number:number:number

A non-negative decimal number of hours followed by a minutes value in the range of zero to 59 inclusive followed by a seconds value in the range of zero to 59 inclusive. For example, `0:00:27`, `0:01:00` or `1:00:00`.

time to live
ttl

The *Time To Live* (TTL) of an IP packet. Valid values are from one (1) through 255, inclusive.

TOS
type of service

The *type of service* is for internet service quality selection. The type of service is specified, along the abstract parameters precedence, delay, throughput, reliability, and cost. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses. The vast majority of IP traffic today uses the default type of service.

RFC references

RFC 827
E. Rosen, *Exterior Gateway Protocol EGP*

RFC 891
D. Mills, *Dcn Local-network Protocols*

RFC 904
D. Mills, *Exterior Gateway Protocol Formal Specification*

RFC 1058
C. Hedrick, *Routing Information Protocol*

RFC 1105
K. Lougheed, Y. Rekhter, *Border Gateway Protocol BGP*

RFC 1163
K. Lougheed, Y. Rekhter, *A Border Gateway Protocol (BGP)*

RFC 1164
J. Honig, D. Katz, M. Mathis, Y. Rekhter, J. Yu,
Application of the Border Gateway Protocol in the Internet

RFC 1227
M. Rose, *SNMP MUX Protocol and MIB*

RFC 1245
J. Moy, *OSPF Protocol Analysis*

RFC 1246
J. Moy, *Experience with the OSPF Protocol*

RFC 1253
F. Baker, R. Coltun, *OSPF Version 2 Management Information Base*

RFC 1256
S. Deering, *ICMP Router Discovery Messages*

RFC 1265
Y. Rekhter, *BGP Protocol Analysis*

RFC 1266
Y. Rekhter, *Experience with the BGP Protocol*

RFC 1267

K. Lougheed, Y. Rekhter, *A Border Gateway Protocol 3 (BGP-3)*

RFC 1268

P. Gross, Y. Rekhter, *Application of the Border Gateway Protocol in the Internet*

RFC 1269

J. Burruss, S. Willis,
Definitions of Managed Objects for the Border Gateway Protocol (v. 3)

RFC 1321

R. Rivest, *The MD-5 Message Digest Algorithm*

RFC 1370

Internet Architecture Board, *Applicability Statement for OSPF*

RFC 1388

G. Malkin, *RIP Version 2 Carrying Additional Information*

RFC 1397

D. Haskin, *Default Route Advertisement In BGP2 And BGP3 Versions Of The Border Gateway Protocol*

RFC 1403

K. Varadhan, *BGP OSPF Interaction*

RFC 1583

J. Moy, *OSPF Version 2*

Configuration File Templates

A

Appendix A contains copies of current templates for GRF configuration files. These copies show how the files appear when opened, before any changes are made.

The appendix includes the following configuration files:

Archiving and saving configuration files

On systems using an RMS node, the **grc** archiving utility saves copies of each of the `/etc` files. Use it to create backups and manage configuration sets.

On the GRF 400, the `/etc` directory resides and is maintained in system RAM. To save `/etc` files between system boots, use the **grwrite** command to save the configuration files to the internal flash disk. **grsnapshot** can be used to archive and name special configurations. **grsite** installs and saves special configuration files for site experimentation and debugging.

GRF configuration files and their uses

This is an alphabetized list of `/etc` GRF configuration files and their application:

`aitmd.conf`	- defines ATMP tunneling parameters
`bridged.conf`	- defines bridge groups and other bridging parameters
`filterd.conf`	- defines system filtering services
`gated.conf`	- enables dynamic routing functions
`grarp.conf`	- maps IP addresses to physical hardware addresses
`grass.conf`	- manages IP services
`gratm.conf`	- configures ATM media cards, including PVCs, logical interfaces for SVCs, peak and sustainable cell rates, UNI signalling, SONET and SDH modes
`grfr.conf`	- configures Frame Relay on HSSI and SONET cards
`grifconfig.conf`	- identifies each logical interface on a media card
`grlamap.conf`	- maps HIPPI logical addresses to media cards
`grppp.conf`	- configures the Point to Point Protocol on HSSI and SONET cards
`grroute.conf`	- sets static routes
`snmpd.conf`	- enables SNMP capabilities
`syslog.conf`	- configures remote logging of log files via **syslogd**

aitmd.conf

This configuration file contains the circuit, foreign agent, and home agent parameters required for the Ascend Tunnel Management Protocol, ATMP. ATMP currently runs on HSSI media cards and supports virtual private networks (VPNs).

aitmd.conf Page 1 of 3

```
# file: /etc/aitmd.conf
#
# This is a sample configuration file for the ATMP home agent server on
# the GRF. It is read by the daemon "aitmd". See the man page for
# aitmd and for aitmd.conf for additional information. By default, the
# aitmd daemon will expect to find its configuration file at # /etc/aitmd.conf
#
# This file contains two kinds of records. First, and simplest, are the
# foreign agent records. These give the IP addresses of ATMP foreign
# agents that are permitted to initiate connections to the home agents.
# Also, they include the password that will be used to authenticate the
# foreign agents to us. The foreign agent must be configured to use
# this password.
#
# Each home network is represented with a different home agent IP
# address on the GRF. For each home network you must configure a name,
# which is what the foreign agent uses to identify the home network.
# "netmask_size" is the number of bits in the netmask used for the
# subnet that the mobile nodes are on. (If you don't know, this can
# be set to zero). This is the subnet in the home network's address # space.
#
# Each home network maps to a different local IP address on the GRF.
# This is the home agent address. This is the address to which the
# foreign agents send to for tunnel negotiation and encapsulated
# traffic. Every home network must have a different IP address. #
# Presently, connections from the GRF back to the home network are only
# supported on the HSSI and ATM-OC3 cards. The home network record
# contains a circuit record which describes the circuit that connects
# back the named home network. The circuit record lists the card
# (0-15), port number (0 or 1), and then two fields for the virtual
# circuit. For frame relay the field "s0" is the DLCI and "s1" should
# be set to zero. For ATM circuits, "s0" is the vpi and "s1" is the # vci.
#
# The hssi/frame relay circuits described in the aitmd configuration
# file must be consistent with pvcatmp entries configured with the
# normal frame relay tools. See the man pages for grfr, grfr.conf,
# and fred for more information on frame relay circuit configuration.
# ATM circuits must be consistent with vc_atmp entries in gratm.conf.
#
# Interfaces
# ----------
# Each circuit used for ATMP must have an interface in grifconfig.conf.
# These interfaces will be used in /etc/grfr.conf for the pvcatmp
# statement, and in /etc/gratm.conf with the PVC where proto=vc_atmp.
# The interfaces in grifconfig.conf for circuits to the ATMP home networks
# should resemble the following. Note that the first three options are
# dashes. Do NOT configure addresses for ATMP interfaces in
# /etc/grifconfig.conf.
#
# gs031 - - - up               # sample interface for HSSI pvcatmp circuit
# ga0288 - - - up              # sample interface for ATM vc_atmp circuit
```

aitmd.conf Page 2 of 3

```
# Syntax
# ------

# Comments begin with the pound sign (#) and continue until the end of
# the line. Most fields are represented with a keyword followed by
# its value. The value must be followed with a semi-colon. Some fields
# are grouped into records. Records begin with a keyword followed
# by a list enclosed withing curly braces {}.
#
# Addresses may be given in dotted decimal notation, such 10.11.12.192,
# or with host names. It is recommended that IP addresses be used.
# Names are more convenient to use and easier to remember, but if the
# DNS name server is unreachable or down, then the aitmd server will not
# be able to convert the name to an address and the configuration will
# fail. Using IP addresses directly makes the system immune to DNS # failures.
#
# Numbers in this sample file are all in normal decimal notation.
# Numeric values like the netmask size, card number, and port number may
# also be entered in hexidecimal by using the '0x' prefix. For example
# 0x4c would be the decimal value 76. If you prefer octal, prefix the
# number with a '0', like 0377 for decimal 255. #

#
# foreign_agent {
#   addr 172.17.1.1;             # IP address of the foreign agent
#   password OurSecret117;       # shared secret.
# }
#
# home_network {
#   name Kansas;                 # text string name. no more than 31 characters
#   netmask_size 16;             # number of bits in the home network netmask
#
#   home_agent_addr 10.2.2.2;    # IP address of the home agent on the grf
#                                # Only one home_network may use this address
#   circuit {
#   card 12;                     # ATM OC-3 card in slot 12
#   port 1;                      # Port 1
#   s0 2;                        # vpi 2
#   s1 88;                       # vci 88
#   }
# }
#
#
# foreign_agent {
#   addr 172.20.5.5;         # IP address of another foreign agent
#   password AnotherSecret;  # shared secret.
# }
#
#
# foreign_agent {
#   addr 172.20.5.100;        # IP address of the foreign agent
#   password TrustNoOne;      # shared secret.
# }
```

aitmd.conf Page 3 of 3

```
#
#
# home_network {
#   name Iowa;                          # text string name. no more than 31 characters
#   netmask_size 24;                    # number of bits in the home network netmask
#
#   home_agent_addr 10.200.7.6;         # IP address of the home agent on the grf
#                                       # Only one home_network may use this address
#   circuit {
#   card 10;                            # HSSI card in slot 10
#   port 0;                             # Port (or link) 0
#   s0 321;                             # DLCI 321
#   s1 0;                               # Unused by frame relay.
#   }
# }
#
```

Figure A-1. Template for the aitmd.conf file

bridged.conf

The `/etc/bridged.conf` configuration file is used to configure bridge groups and other bridging parameters.

bridged.conf Page 1 of 4

```
#   NetStar $Id: bridged.conf,v 1.17.10.1 1997/11/18 23:27:21 pdm Exp $
#
# Configuration file for Bridge Daemon (bridged).
#
#   Note: bridged will not start if it finds an error while
#    trying to parse this file. Use the "-d" option on the
#    command line with bridged to find proximity of the offending line.
#

bridge_group bg0 {
    #
    # The main reason we need to configure this stuff is to
    # specify which ports are part of a bridge group.

    # Declare all the ports in this group on a single
    # line if you don't need to set anything special
    # for the port. This is the normal case.
    #
    # multiple lines are ok.
    #
    # eg. port gf070;
    #     port gf041 gf072;
    #     port gf080;
    #

    port gf090 gf091;

    #
    # If you need to set specific values for a port in this
    # bridge group, then use the structure below..
    #
    # port gf040 {
        #
        #
        # priority : port priority allows the network manager to
        #   influence the choice of port when a bridge has
        #   two ports connected in a loop.
        #
        #   priority 5;
        #
```

bridged.conf Page 2 of 4

```
        # valid states are : blocking, listening, learning and
        #   forwarding. the default start state is blocking.
        #

        #   state blocking;
        #
        # root_path_cost : This is the cost to be added to the root
        #   path cost field in a configuration message received
        #   on this port in order to determine the cost of the
        #   path to the root through this port. This value is
        #   individually settable on each port.
        #   Setting this value to be large on a particular port
        #   makes the LAN reached through that port more likely
        #   to be a lead or at least low in the spanning tree.
        #   The closer a LAN is to being a leaf in the tree, the
        #   less through traffic it will be asked to carry. A
        #   LAN would be a candidate for having a large path
        #   cost if it has a lower bandwidth ot if someone wants
        #   to minimize unnecessary traffic on it. A better
        #   description is possibly link cost or port cost.
        #

        #   root_path_cost 5;
        #
        # Forward packets with the following destination addresses
        # through this port.
        #
        #   forward 00:a0:24:2a:50:e6 00:a0:24:2a:50:e7;
        #
    # };
    # port gf041 {
        #   state blocking;
        #   root_path_cost 6;
        #   priority 5;
    # };

    #
    # Configuration and tuning parameters that
    # govern how a bridge group functions.
    #
    # priority: This is used to create the bridge ID for
    #   this bridge. this along with the MAC address of one
    #   of the ports in the group is used to set the bridge ID.
    #   This value allows the network manager to influence the
    #   choice of root bridge and the designated bridge. It is
    #   appended as the most significant portion of a bridge ID.
    #   A lower numerical value for bridge priority makes the
    #   bridge more likely to become the root.
    #
    #   priority 128;
    #
    #
    # hello_time: Interval between the transmission of configuration
    #   BPDUs by a bridge that is attempting to become the root
    #   bridge or is root bridge.
    #
```

bridged.conf Page 3 of 4

```
#   It is the timer that elapses
#   between generation of configuration messages by a bridge
#   that assumes itself to be the root.
#   Shortening this time will make the protocol more
#   robust, in case the probability of loss of configuration
#   messages is high. Lengthening the timer lowers the overhead
#   of the alogorithm (because the interval between transmission
#   of configuration messages will be larger).
#
#   The recommended time is 2 sec.
#
#   hello_time 2 seconds;

# forward_delay: The time value advertised by this
#   bridge for deciding the time delay that a port must
#   spend in the listening and learning states.
#
#   This parameter temporarily prevents a bridge from starting
#   to forward data packets to and from a link until news of
#   a topology change has spread to all parts of a bridged
#   network. This shousl give all links that need to be turned
#   off in the new topology time to do so before new links are
#   turned on.
#


#   Setting the forward delay too small would result in temporary
#   loops as the spannign tree algorithm converges. Setting this
#   value too large results in longer partitions after the
#   spanning tree reconfigures.
#
#   The recommended value is 15 sec.

forward_delay 15 seconds;

#
# maximum_age: This is the time value advertised by this bridge for
#   deciding whether to discard spanning tree frames based on
#   message age.
#
#   If the selected max_age value is too small, then occasionally,
#   the spanning tree will reconfigure unnecessarily, possibly
#   causing temporary loss of connectivity in the network. If the
#   selected value is too large, the network will take longer then
#   necessary to adjust to a new spanning tree after a topological
#   event such as restarting or crashing of a bridge or link.
#
#   The recommended value is 20 sec.
#
#   maximum_age  20 seconds;
```

bridged.conf Page 4 of 4

```
    #
    # route_maximum_age: This parameter determines how often routes
    #   will be aged out of the learnt route table.
    #
    #   Default : 300 seconds
    #
    #   route_maximum_age 300 seconds;
    #
    # And last, hardwiring the forwarding table with
    # specific MAC addresses to discard.
    #
    # Sink packets with the following destination
    # addresses.
    #
    #   discard 00:40:0b:0c:95:60 00:40:0b:0c:95:6a;
    #
    # Disable Spanning Tree for the group
    #
    #   spanning_tree disabled
} ;


#
# To create a bridge group with no ports in it, use the
# following NULL declaration:
#
#   bridge_group bg0 {
#           ;
#   };

#debug_level 5;  # traces all events of level NOTICE and above
```

Figure A-2.   Template of bridged.conf file

filterd.conf file

In `filterd.conf`, you can name a filter and specify the filter's rules, and assign (bind) the filter to a particular media card. The filter function is enabled after you copy the template file, `/etc/filterd.conf.template`, to `/etc/filterd.conf` and edit and save the file.

filterd.conf Page 1 of 3

```
# Template file to give an example of how filtering is setup.
#
# The first step is to define filters that you want to have.  These are
# general items that can be applied to more than one media card/interface or
# don't have to be applied at all.  If you are not using a filter,
# however, comment it out using /* filter.... */ to allow for faster
# load time.
#
#
# The following is a a fairly complex filter example that denies access to the
# world but lets in some "things".  A definition of each of these "things" is
# included in a comment by each rule.
#
# In GR filtering lingo, the following is a "filter":
#

/* ------
filter example_in_1 {
    implicit deny;# if no rules matched, filter DENIES

    #
    # Make our first match rule.  In this case allow packets from
    # ports > 1023 and returning packets in established TCP connections.
    #
    # In GR filtering lingo, this is a "rule".
    permit {
        ipv4protocol tcp {
            established;
            port gt 1023;
        }
        ipv4protocol udp {
            port gt 1023;
        }
    }

    #
    # Ok, now lets do something to allow people to talk to our
    # mail server.
    #
    permit {
        to 222.222.222.1 0.0.0.0;
        ipv4protocol tcp {
            port 25;
        }
    }
```

filterd.conf Page 2 of 3

```
    #
    # Let the consultant from somehost.somewhere.com into one of our
    # networks.
    #
    permit {
        from 221.222.222.32 0.0.0.0;
        to 222.222.222.0 0.0.0.255;
    }
}
#
#
# The following is a second filter, which we will stick on our transmit side
# of our "internal" network. The filter prevents someone from dropping packets
# from the outside that look like they come from the inside.
#
# Note that it may make more sense to place this on the upstream side
# of all the "unsafe" interfaces instead of on the downstream side.  This
# is especially true if there are more than one interface on the "safe"
# side that move traffic amongst themselves.  Putting the filter on the
# downstream side in this case causes safe traffic to take the performance
# hit of blocking traffic on the upstream side.
#

filter spoof_block {
    # By default, let everything through.  Our upstream side will catch
    # the stuff it wants to weed out.
    implicit permit;

    # our "internal" network is 222.222.222.*.
    deny {
        from 222.222.222.0 0.0.0.255;
        to 222.222.222.0 0.0.0.255;
    }
}

#
# And now, an unusual filter to show how some other things work, but
# would never actually be created.
#
filter weird_filter {
    implicit deny;

    permit {
        from 128.101.101.101 0.0.0.0;# from this one
        from 128.101.101.102 0.0.0.0;# OR from this one
        to 220.220.220.0 0.0.0.255;# AND to this one
        ipv4protocol icmp;# ICMPs allowed
        ipv4protocol tcp {
            # note that allow port 0, even though it is not
            # a valid port.  By including it, range checking
            # becomes more efficient in many cases.
            port 0..3, 10..20, gt 1023;
        }
        ipv4protocol udp;# all udp through
    }
}
```

filterd.conf Page 3 of 3

```
# Once all the filters are declared, we can declare what is called a "binding"
# This means attaching the above filters to interfaces.
#

media fddi 11 {
        # ok, what filter?
        bind example_in_1 {
                vlif 0;                 # do the first interface
                direction in;           # inbound traffic only
                action filter;          # route/don't route
        }

        #
        # and we have another interface that we are going to pretend
        # routes to our internal net.
        bind spoof_block {
                vlif 1;                 # the other interface (DAS)
                direction out;          # outbound traffic only
                action filter;
        }
}

#
# And another card for our _other_ "internal" network.  We attach the
# weird filter to it.
media atm 3 {
        bind weird_filter {
                vlif 0..255;            # all interfaces
                direction in;           # in
                direction out;          # AND outbound
                action filter;
        }
}
```

Figure A-3. Template for the filterd.conf file

gated.conf components

The next pages show many of the components available for use in a `gated.conf` file.

gated.conf Page 1 of 10

```
# Trace option statement
   traceoptions ["trace_file" [replace] [ size size[k|m] files files ]]
           [control_options] trace_options [except trace_options] ;
   traceoptions none ;

# Option statements
options
    [ nosend ]
    [ noresolv ]
    [ gendefault [ precedence precedence ] [ gateway gateway] ]
    [ syslog [ upto ] log_level ]
    [ mark time ]
    ;

# Interface statements
interfaces {
    options
        [ strictinterfaces ]
        [ scaninterval time ]
        ;
    interface interface_list
        [ precedence precedence ]
        [ down precedence precedence ]
        [ passive ]
        [ simplex ]
        [ reject ]
        [ blackhole ]
        ;
    define address
        [ broadcast address ] | [ pointtopoint address ]
        [ netmask mask ]
        [ multicast ]
        ;
} ;
# Definition statements
autonomoussystem autonomous_system [ loops number ] ;
routerid host ;
confederation confederation ;
routing-domain rdi ;
martians {
        host host [ allow ] ;
        network [ allow ] ;
        network mask mask [ allow ] ;
        network ( masklen | / ) number [ allow ] ;
        default [ allow ] ;
    } ;

# RIP statements
rip yes | no | on | off [ {
    broadcast ;
    nobroadcast ;
    nocheckzero ;
    preference preference ;
    defaultmetric metric ;
    query authentication [none | ([simple|md5] password)] ;
```

gated.conf Page 2 of 10

```
# RIP statement, continued
 interface interface_list
        [noripin] | [ripin]
        [noripout] | [ripout]
        [metricin metric]
        [metricout metric]
        [version 1]|[version 2 [multicast|broadcast]]
        [[secondary] authentication [none| ([simple|md5] password)]]
    trustedgateways gateway_list ;
    sourcegateways gateway_list ;
    traceoptions trace_options ;
} ] ;


# OSPF statement
ospf yes | no | on | off [ {
    defaults {
        precedence precedence ;
        cost cost ;
        tag [ as ] tag ;
        type 1 | 2 ;
        inherit-metric ;
    } ;
    exportlimit routes ;
    exportinterval time ;
    traceoptions trace_options ;
    monitorauthkey authkey ;
    monitorauth none | ( [ simple | md5 ] authkey ) ;
    backbone | ( area area ) {
        authtype  none | simple ;
        stub [ cost cost] ;
        networks {
            network [ restrict ] ;
            network mask mask [ restrict ] ;
            network ( masklen | / ) number [ restrict ] ;
            host host [ restrict ] ;
        } ;
        stubhosts {
            host  cost cost ;
        } ;
        interface interface_list; [cost cost ] {
            interface_parameters
        } ;
        interface interface_list nonbroadcast [cost cost ] {
            pollinterval time ;
            routers {
                gateway [ eligible ] ;
            } ;
            interface_parameters
        } ;
        Backbone only:
        virtuallink neighborid router_id transitarea area {
            interface_parameters
        } ;
    } ;
} ] ;
```

gated.conf Page 3 of 10

```
# IS-IS statement
       isis no | ip {
          level 1|2 ;
          [traceoptions <isis_traceoptions> ;]
          [systemid <string> ;]
          [area <string> ;]
          [set <isis_parm> value ;]
          circuit|interface <interface-name>
              metric [level 1|2] metric
              priority [level 1|2] priority pointopoint ;
          [ipreachability level (1|2) (internal|external|summary)
              ipaddr netmask [metric metric;]]
       } ;

# EGP statement
    egp yes | no | on | off
    [ {
        preference preference ;
        defaultmetric metric ;
        packetsize number ;
        traceoptions trace_options ;
        group
            [ peeras autonomous_system ]
            [ localas autonomous_system ]
            [ maxup number ]
        {
            neighbor host
                [ metricout metric ]
                [ preference preference ]
                [ preference2 preference ]
                [ ttl ttl ]
                [ nogendefault ]
                [ importdefault ]
                [ exportdefault ]
                [ gateway gateway ]
                [ lcladdr local_address ]
                [ sourcenet network ]
                [ minhello | p1 time ]
                [ minpoll | p2 time ]
                [ traceoptions trace_options ]
                ;
       } ;
    } ] ;

# BGP statement
    bgp yes | no | on | off
    {
        protocol-precedence precedence ;
        allow bad community;        \
        defaultmetric metric ;
        traceoptions trace_options ;
        [ clusterid host ; ]
     [ group type
             ( external peeras autonomous_system
                    [ ignorefirstashop ]  [ subgroup integer ]
                    [ med ] )
```

gated.conf Page 4 of 10

```
# EGP statement, continued
           | ( internal peeras autonomous_system
                    [ ignorefirstashop ]
                    [ lcladdr local_address ]
                    [ outdelay time ]
                    [ metricout metric ]
                    [ reflector-client [ no-client-reflect ] ]
                    [ subgroup integer ]

          | ( routing peeras autonomous_system proto proto_list
                    interface interface_list
                    [ ignorefirstashop ]
                    [ lcladdr local_address ]
                    [ outdelay time ]
                    [ metricout metric ]
                    [ reflector-client [ no-client-reflect ] ]
                    [ subgroup integer ]

          | ( confed peeras autonomous_system proto proto_list
                    interface interface_list
                    [ ignorefirstashop ]
                    [ lcladdr local_address ]
                    [ outdelay time ]
                    [ metricout metric ]
                    [ reflector-client [ no-client-reflect ] ]
                    [ subgroup integer ]

          | ( test peeras autonomous_system )  ]
            [ aspath-opt ]
            {
              [  allow {
                    network
                    network mask mask
                    network ( masklen | / ) number
                    all
                    host host  ]
                 } ;
                 peer host
                    [ metricout metric ]
                    [ localas autonomous_system ]
                    [ ignorefirstashop ]
                    [ nogendefault ]
                    [ gateway gateway ]
                    [ nexthopself ]
                    [ protocol-precedence precedence ]
                    [ preference preference ]
                    [ lcladdr local_address ]
                    [ holdtime time ]
                    [ version number ]
                    [ passive ]
                    [ sendbuffer number ]
                    [ recvbuffer number ]
                    [ outdelay time ]
                     [ keep [ all | none ] ]
                    [ show-warnings ]
                    [ noaggregatorid ]
                    [ keepalivesalways ]
```

gated.conf Page 5 of 10

```
# BGP statement, continued
                    [ v3asloopokay ]
                    [ nov4asloop ]
                    [ ascount count ]
                    [ throttle count ]
                    [ allow bad routerid ]
                    [ logupdown ]
                    [ ttl ttl ]
                    [ traceoptions trace_options ]
                    ;
            } ;
        } ;

# Weighted route dampening statement
   dampen-flap {
      [suppress-above metric;
      reuse-below metric;
      max-flap metric;
      unreach-decay time;
      reach-decay  time;
      keep-history  time; ]
   };

# ICMP statement
   icmp {
       traceoptions trace_options ;
   }

# Router discovery statement
   routerdiscovery server yes | no | on | off [ {
       traceoptions trace_options ;
       interface interface_list
           [ minadvinterval time ] |
           [ maxadvinterval time ] |
           [ lifetime time ]
           ;
       address interface_list
           [ advertise ] | [ ignore ] |
           [ broadcast ] | [ multicast ] |
           [ ineligible ] | [ preference preference ]
           ;
   } ] ;

# Router discovery client statement
   routerdiscovery client yes | no | on | off [ {
       traceoptions trace_options ;
       preference preference ;
       interface interface_list
           [ enable ] | [ disable ]
           [ multicast ]
           [ quiet ] | [ solicit ]
           ;
   } ] ;
```

gated.conf Page 6 of 10

```
# Kernel statement
   kernel {
       options
           [ nochange ]
           [ noflushatexit ]
           ;
       routes number ;
       flash
           [ limit number ]
           [ type interface | interior | all ]
           ;
       background
           [ limit number ]
           [ priority flash | higher | lower ]
          ;
       traceoptions trace_options ;
   } ;

# Static statement
   static {
       ( host host ) | default |
       ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
           gateway gateway_list
           [ interface interface_list ]
           [ preference preference ]
           [ retain ]
           [ reject ]
           [ blackhole ]
           [ noinstall ] ;
       ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
           interface interface
           [ preference preference ]
           [ retain ]
           [ reject ]
           [ blackhole ]
           [ noinstall ] ;
   } ;

# Filtering statements
   network mask mask [ exact | refines | between number and number ]
   network masklen |/ number [ exact | refines | between number and number ]
   all
   default
   host host

# Autonomous system matching statement
   aspath aspath_regexp origin any | ( [ igp ] [egp ] [ incomplete ] )

# AS path attribute statement
   aspath-opt {
       [ community autonomous_system : community-id | community-id ]
       [ community no-export | no-advertise | no-export-subconfed | none ]
   }
   mod-aspath {
       [ community autonomous_system : community-id |community-id ]
       [ community no-export | no-advertise | no-export-subconfed ]
       [ comm-split autonomous_system community-id ]
   }
```

gated.conf Page 7 of 10

```
# Import from BGP and EGP statements
   import proto bgp | egp autonomoussystem autonomous_system
       [ subgroup integer ]  [ aspath-opt ] restrict ;
   import proto bgp | egp autonomoussystem autonomous_system
       [ subgroup integer ] [ aspath-opt ] [ preference preference ] {
       route_filter  [ restrict | ( precedence  precedence )
   | ( preference preference ) | ( localpref preference ) ] ;
   } ;

   import proto bgp aspath aspath_regexp
       origin any | ( [ igp ] [egp ] [ incomplete ] )
       [ aspath-opt ] restrict ;
   import proto bgp aspath aspath_regexp
       origin any | ( [ igp ] [egp ] [ incomplete ] )
       [ aspath-opt ] [ preference preference ] {
       route_filter [ restrict | ( precedence  precedence )
   | ( preference preference ) | ( localpref preference ) ] ;
   } ;

# Import from RIP, Hello, and Redirect statements
   import proto rip | hello | redirect
       [ ( interface interface_list ) | (gateway gateway_list ) ]
       restrict ;
   import proto rip | hello | redirect
       [ ( interface interface_list ) | (gateway gateway_list ) ]
       [ preference preference ] {
       route_filter  [ restrict | ( precedence  precedence )
   | ( preference preference ) | ( localpref preference ) ] ;
   } ;

# Import from OSPF statements
   import proto ospfase [ tag ospf_tag ] restrict ;
   import proto ospfase [ tag ospf_tag ]
       [ preference preference ] {
       route_filter  [ restrict | ( precedence  precedence )
   | ( preference preference ) | ( localpref preference ) ] ;
   } ;



# Export to BGP and EGP statements
   export proto bgp | egp as autonomous system
       restrict ;
   export proto bgp | egp as autonomous system [ mod-aspath]
       [ metric metric  | localpref preference]  [ subgroup integer] {
       export_list ;
   } ;

# Export to RIP and Hello statements
   export proto rip | hello
       [ ( interface interface_list ) | (gateway gateway_list ) ]
       restrict ;
   export proto rip | hello
       [ ( interface interface_list ) | (gateway gateway_list ) ]
       [ metric metric ] {
       export_list ;
   } ;
```

gated.conf Page 8 of 10

```
# Export to OSPF statements
   export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
       restrict ;
   export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
       [ metric metric ] {
       export_list ;
   } ;


# Exporting BGP and EGP routes

   proto bgp | egp autonomoussystem autonomous_system   [subgroup integer ]
       restrict ;
   proto bgp | egp autonomoussystem autonomous_system  [subgroup integer ]
       [ metric metric  | localpref preference ] {
       route_filter [ restrict | ( metric metric ) ] ;
   } ;

# Exporting RIP and HELLO routes

   proto rip | hello
       [ ( interface interface_list ) | (gateway gateway_list ) ]
       restrict ;
   proto rip | hello
       [ ( interface interface_list ) | (gateway gateway_list ) ]
       [ metric metric ] {
       route_filter [ restrict | ( metric metric ) ] ;
   } ;

# Exporting OSPF routes
   proto ospf | ospfase restrict ;
   proto ospf | ospfase [ metric metric ] {
       route_filter [ restrict | ( metric metric ) ] ;
   } ;


# Exporting routes from non-routing protocol statement
   proto direct | static | kernel
       [ (interface interface_list ) ]
       restrict ;
   proto direct | static | kernel
       [ (interface interface_list ) ]
       [ metric metric ] {
       route_filter [ restrict | ( metric metric ) ] ;
   } ;

# Exporting bt AS path
   proto proto | all aspath aspath_regexp
       origin any | ( [ igp ] [egp ] [ incomplete ] )
       restrict ;
   proto proto | all aspath aspath_regexp
       origin any | ( [ igp ] [egp ] [ incomplete ] )
       [ metric metric  | localpref preference ] {
       route_filter [ restrict | ( metric metric ) ] ;
   } ;
```

gated.conf Page 9 of 10

```
# Exporting by route tag
   proto proto | all tag tag restrict ;
   proto proto | all tag tag
       [ metric metric ] {
       route_filter [ restrict | ( metric metric ) ] ;
   } ;

# Aggregation and generation statement
   aggregate default
       | ( network [ ( mask mask ) | ( ( masklen number | / ) number ) ] )
       [ preference preference ] [ brief | truncate ] {
      proto [ all | direct | static | kernel | aggregate | proto ]
           [ ( as autonomous_system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
           restrict ;
       proto [ all | direct | static | kernel | aggregate | proto ]
           [ ( as autonomous_system ) | ( tag tag )
               | ( aspath aspath_regexp ) ]
           [ preference preference ] {
           route_filter [ restrict | ( preference preference ) ] ;
       } ;
   } ;
   generate default
       | ( network [ ( mask mask ) | ( ( masklen number | / ) number ) ) ] )
       [ preference preference ] {
           [ ( as autonomous_system ) | ( tag tag )
               | ( aspath aspath_regexp ) ]
           restrict ;
       proto [ all | direct | static | kernel | aggregate | proto ]
           [ ( as autonomous_system ) | ( tag tag )
               | ( aspath aspath_regexp ) ]
           [ preference preference ] {
           route_filter [ restrict | ( preference preference ) ] ;
       } ;
   } ;


# Directive statement
   %directory "directory"

# Include statement
   %include "filename"
```

Figure A-4. Components configurable in a gated.conf file

grarp.conf file

The `/etc/grarp.conf` file is set up to configure general address resolution for HIPPI, ATM, and FDDI media cards. It maps IP addresses to physical hardware addresses in support of ARP functions.

`grarp.conf` can also be used to manually manipulate the ARP cache to resolve a temporary network irregularity. This type of problem occurs when, for example, a destination's file does not properly respond to ARP requests, or when its hardware address has recently changed. When an ATM destination does not respond to Inverse ARP requests, `grarp.conf` supplies the IP address information that should have come from the InARP reply.

Here is the `grarp.conf` file format with sample entries:

```
[ifname]      hostname        phys_addr   [temp]   [pub]   [trail]

gh010         192.0.2.1       0x          temp
gf023         192.0.99.1      0x          pub trail
ga0364        192.0.130.1     0x
```

ifname

`ifname` is the interface name in the form "`gx0yz`."
The interface name defines the physical location in the chassis of each configured logical interface. It is used in the `grifconfig.conf` file to identify each logical interface.

hostname

`hostname` is the IP address or name of the host to which the address is to be mapped.

phys_addr

The third entry, `phys_addr`, is the remote system's hardware address. The format of this entry is specific to each type of media.

For 100Base-T (Fast Ethernet) or FDDI, use the 48-bit MAC address (in hex):

```
xx:xx:xx:xx:xx:xx
```

For HIPPI, use the 32-bit I-field, an unsigned integer in C language convention (prefixed with "0x" for hexadecimal, "0" for octal, or a non-zero digit for decimal).

For ATM, there are three options:

– the NSAP address in 20 hexadecimal bytes, two digits each, optionally separated by periods:

```
xx.xx.xx ... xx.xx.xx
```

Multiple NSAP addresses are separated by plus signs (+).

– for PVC connections, the VPI/VCI values are in decimal integers separated by a slash:

```
vp/vc
```

– in CCITT E.164-style address

To configure ARP on the ATM media card, you must specify addresses of local ARP servers in the `gratm.conf` file in addition to setting addresses in `grarp.conf`.

See the **grarp** command description or the **grarp** man page for more information.

This is the template for `grarp.conf`:

grarp.conf Page 1 of 1

```
#  NetStar $Id: grarp.conf,v 1.2.22.1 1997/05/09 17:35:00 jim Exp $
#
# Template grarp.conf file.
#
#   This file contains any hardwired IP-address-to-hardware-address
#   mappings you may want for GigaRouter interfaces.  It is especially
*   important for HIPPI, whose ARP tables are manually configured.
#   For HIPPI, this file replaces the "gria" command.
#
# File syntax:
#
# [ifname]  host    hwaddr   [temp]  [pub]  [trail]
#
# [ifname]   If given, this is the interface name as you would find it
#   in column 1 of the netstat -i output.
#
# host    Hostname or IP address of the remote system
#
# hwaddr   Hardware address of the remobe system in one of the
#   following formats:
#
#       48-bit MAC address for
#       Ethernet or FDDI:  xx:xx:xx:xx:xx:xx
#                 where 'xx' are hexadecimal digits.
#
#       32-bit I-field for
#       HIPPI:  C-language syntax for a 32-bit constant
#           Example:  0x03000555 for logical address x'555.
#
#       20-byte NSAP address or VPI/VCI for
#       ATM:   vp/vc    VPI/VCI for PVCs
#              where 'vp' and 'vc' are decimal integers
#           xx.xx.xx. . .xx.xx.xx.xx   NSAP address
#              where 'xx' are hexadecimal digits.
#
#
#############   Add your configuration following this line   ############
```

Figure A-5. Template for the grarp.conf file

grass.conf file

The **grass** command enables/disables internal IP service settings in the operating system. The `/etc/grass.conf` file should not be edited by the user. Use **grass** to display the current status of all managed services as viewed by their configuration files. Services are listed in the `/etc/services` file.

This is the template for `grass.conf`, double-columned because of its length:

grass.conf Page 1 of 3

```
# NOTE: grass.conf is not intended to be modified by
the customer.  Any
# modifications will be overwritten at upgrade time.

#
service ftp {
        file /etc/inetd.conf;
        match ftp[[:space:]]*stream[[:space:]]*tcp;
        script /etc/grass/inetd.grass.pl;
}

service telnet {
        file /etc/inetd.conf;
        match telnet[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service shell {
        file /etc/inetd.conf;
        match ^[#[:space:]]*shell[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service login {
        file /etc/inetd.conf;
        match ^[#[:space:]]*login[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service exec {
        file /etc/inetd.conf;
        match exec[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service uucpd {
        file /etc/inetd.conf;
        match uucpd;
        script /etc/grass/inetd.grass.pl;
}

service finger {
        file /etc/inetd.conf;
        match finger;
        script /etc/grass/inetd.grass.pl;
}
service tftp {

        file /etc/inetd.conf;
        match tftp;
        script /etc/grass/inetd.grass.pl;
}
```

```
service comsat {
        file /etc/inetd.conf;
        match comsat;
        script /etc/grass/inetd.grass.pl;
}

service ntalk {
        file /etc/inetd.conf;
        match ntalk;
        script /etc/grass/inetd.grass.pl;
}

service pop {
        file /etc/inetd.conf;
        match pop;
        script /etc/grass/inetd.grass.pl;
}

service ident {
        file /etc/inetd.conf;
        match ident;
        script /etc/grass/inetd.grass.pl;
}

service bootp {
        file /etc/inetd.conf;
        match bootp;
        script /etc/grass/bootp.grass.pl;
}
#
# echo, discard, chargen, daytime, and
time all have both tcp and udp
#      components.  Add a _tcp or _udp for
each sub-type.
service echo_tcp {
        file /etc/inetd.conf;
        match echo[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service discard_tcp {
        file /etc/inetd.conf;
        match discard[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service chargen_tcp {
        file /etc/inetd.conf;
        match chargen[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}
```

grass.conf Page 2 of 3

```
service daytime_tcp {
        file /etc/inetd.conf;
        match daytime[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service time_tcp {
        file /etc/inetd.conf;
        match
^[#[:space:]]*time[[:space:]]*stream;
        script /etc/grass/inetd.grass.pl;
}

service echo_udp {
        file /etc/inetd.conf;
        match echo[[:space:]]*dgram;
        script /etc/grass/inetd.grass.pl;
}

service discard_udp {
        file /etc/inetd.conf;
        match discard[[:space:]]*dgram;
        script /etc/grass/inetd.grass.pl;
}

service chargen_udp {
        file /etc/inetd.conf;
        match chargen[[:space:]]*dgram;
        script /etc/grass/inetd.grass.pl;
}

service daytime_udp {
        file /etc/inetd.conf;
        match daytime[[:space:]]*dgram;
        script /etc/grass/inetd.grass.pl;
}

service time_udp {
        file /etc/inetd.conf;
        match
^[#[:space:]]*time[[:space:]]*dgram;
        script /etc/grass/inetd.grass.pl;
}

service tcpmux {
        file /etc/inetd.conf;
        match tcpmux;
        script /etc/grass/inetd.grass.pl;
}


service amanda {
        file /etc/inetd.conf;
        match amanda;
        script /etc/grass/inetd.grass.pl;
}

service klogin {
        file /etc/inetd.conf;
        match ^[#[:space:]]*klogin;
        script /etc/grass/inetd.grass.pl;
}

service eklogin {
        file /etc/inetd.conf;
        match eklogin;
        script /etc/grass/inetd.grass.pl;
}
service kshell {
        file /etc/inetd.conf;
        match ^[#[:space:]]*kshell;
        script /etc/grass/inetd.grass.pl;
}

service krbupdate {
        file /etc/inetd.conf;
        match krbupdate;
        script /etc/grass/inetd.grass.pl;
}

service kpasswd {
        file /etc/inetd.conf;
        match kpasswd;
        script /etc/grass/inetd.grass.pl;
}
#
# And from rc
#
service nfsd {
        file /etc/rc;
        match ' nfsd';
        script /etc/grass/dogeneric.grass.pl;
}

service portmap{
        file /etc/rc;
        match portmap;
        script /etc/grass/dogeneric.grass.pl;
}

service named {
        file /etc/rc;
        match ' named';
        script /etc/grass/dogeneric.grass.pl;
}

service mountd{
        file /etc/rc;
        match ' mountd';
        script /etc/grass/dogeneric.grass.pl;
}

service xntpd {
        file /etc/rc;
        match ' xntpd';
        script /etc/grass/dogeneric.grass.pl;
}

service nfsiod {
        file /etc/rc;
        match ' nfsiod';
        script /etc/grass/dogeneric.grass.pl;
}

service timed {
        file /etc/rc;
        match ' timed';
        script /etc/grass/dogeneric.grass.pl;
}
```

grass.conf Page 3 of 3

```
service rwhod {
        file /etc/rc;
        match " rwhod ;
        script /etc/grass/dogeneric.grass.pl;
}


service rstatd{
        file /etc/rc;
        match ' rstatd';
        script /etc/grass/dogeneric.grass.pl;
}

service lpd {
        file /etc/rc;
        match ' printer';
        script /etc/grass/dogeneric.grass.pl;
}

service sendmail {
        file /etc/rc;
        match ' sendmail';
        script /etc/grass/dogeneric.grass.pl;
}

service amd {
        file /etc/rc;
        match ' amd';
        script /etc/grass/dogeneric.grass.pl;
}

service httpd {
        file /etc/rc;
        match ' httpd';
        script /etc/grass/dogeneric.grass.pl;
}

service nmbd {
        file /etc/rc;
        match ' nmbd';
        script /etc/grass/dogeneric.grass.pl;
}

service smbd {
        file /etc/rc;
        match ' smbd';
        script /etc/grass/dogeneric.grass.pl;
}

service pcnfsd {
        file /etc/rc.local;
        match " pcnfsd";
        script /etc/grass/dogeneric.grass.pl;
}
```

```
group all {
        service ftp;
        service telnet;
        service shell;
        service login;
        service exec;
        service uucpd;
        service finger;
        service tftp;
        service comsat;
        service ntalk;
        service pop;
        service ident;
        service bootp;
        service echo_tcp;
        service discard_tcp;
        service chargen_tcp;
        service daytime_tcp;
        service time_tcp;
        service echo_udp;
        service discard_udp;
        service chargen_udp;
        service daytime_udp;
        service time_udp;
        service tcpmux;
        service amanda;
        service klogin;
        service eklogin;
        service kshell;
        service krbupdate;
        service kpasswd;
        service nfsd;
        service portmap;
        service named;
        service mountd;
        service xntpd;
        service timed;
        service nfsiod;
        service rwhod;
        service rstatd;
        service lpd;
        service sendmail;
        service amd;
        service httpd;
        service nmbd;
        service smbd;
        service pcnfsd;
}
group diagnostic {
        service echo_tcp;
        service discard_tcp;
        service chargen_tcp;
        service daytime_tcp;
        service time_tcp;
        service echo_udp;
        service discard_udp;
        service chargen_udp;
        service daytime_udp;
        service time_udp;
}
```

Figure A-6. Template for the grass.conf file

gratm.conf file

The `/etc/gratm.conf` file maps IP addresses to physical hardware addresses for ATM OC-3c/Q and ATM OC-12c media cards. Parameters to configure ATMP tunneling and encapsulated bridging are also provided.

The first entry for each interface specifies a signalling protocol for switched virtual circuits (SVCs) configured on that interface. Legal protocols are `UNI 3.0`, `UNI 3.1`, or `none`. When the protocol field is left blank, UNI 3.0 signalling is used by default.

The second entry for an interface sets rate queues to which switched and physical virtual circuits (SVCs and PVCs) can be assigned. Rates are described in kilobits per second and are a way to allocate available bandwidth.

Example

```
This example sets up rate queues for both physical interfaces on the ATM
media card in slot 9:

# cage   card   interface       protocol
#       queue                   rate
#
# physical interface 0 signaling
0       9         0             NONE
# physical interface 0 queues
        0                       155000
        1                        85000
        2                        55000
        3                        25000
        4                        15000
# physical interface 1 signaling
0       9         1             NONE
# physical interface 1 queues
        0                       155000
        1                       100000
        2                       115000
        3                        65000
        4                        55000
        5                        35000
        6                        25000
        7                        5000
#
```

The `gratm.conf` file template is on the next five pages.

gratm.conf Page 1 of 5

```
# NetStar $Id: gratm.conf,v 1.11.2.1 1997/11/25 16:21:58 pdm Exp $
#
# gratm.conf - GigaRouter ATM Configuration File
#
#
# This file is used to configure GigaRouter ATM interfaces.
# Statements in this file are used to configure ATM PVC's,
# signalling protocols, arp services, and traffic shapes.
#
# gratm(8) uses this file as input when it is run by grinchd(8)
# whenever an ATM media card boots to configure the card.
#

#
# gratm.conf is divided into five sections:
#
# The Service section is where ATM ARP services are defined (entries
#           defined in this section are referenced from the Interface section
#           of this file to define which ARP service an interface should use).
#
# The Traffic Shaping section is where traffic shapes are defined (entries
#           defined in this section are referenced from the Interface and PVC
#           sections of this file to define which traffic shapes interfaces
#           and PVC's should use).
#
# The Signalling section is where the signalling protocol to be used
#           by a physical interface to establish Switched Virtual Circuits
#           is specified.
#
# The Interfaces section is where per-logical-interface parameters
#           such as ARP services and Traffic shapes are bound to specific
#           logical interfaces.
#
# The PVC section is where Permanent Virtual Circuits are defined,
#           using traffic shapes defined in the Traffic Shaping section,
#           along with other parameters specific to PVC configuration.
#


#
# Notes on the format of this file:
#
# Comments follow the Bourne Shell style (all characters following a #
# on a line are ignored).
#
# Statements in this file are separated by newlines.  A statement may
# span multiple lines by ending each incomplete line of the statement
# with a '\' character.  Example:
#
# Traffic_Shape name=high_speed peak=15000   # this is a statement
#
# Service name=bc0 type=bcast addr=198.174.11.1 \  # this is also a statement
#               addr=198.176.11.1
#
# User-defined names for Traffic_Shape's and Service's must be defined
# before they are used, ie., the definition of a traffic shape must
# precede its use in an Interface or PVC specification, and the
# definition of a Service must precede the use of the name of that
# service in defining any Interfaces.
# ARP Service info
#
# Lines beginning with the keyword "Service" define virtual "services" which
# may or may not be present on an ATM network attached to a GigaRouter.
#
# Each Service entry the ATM configuration file has the following format:
#
# Service name=value  type=arp|bcast   addr=value   [addr=value ...]
#
# The "name" field is a unique name to identify this ATM service
```

gratm.conf Page 2 of 5

```
# The "type" field specifies the type of ATM service being configured.
# and how the address argument(s) which follow are interpreted.
#
#   type=arp     Indicates an ARP service.  One to three "addr" field(s)
#                      must follow, defining NSAP addresses of ARP
#                      services on the attached ATM network.
#
#                      A logical interface using this "Service" entry
#                      will connect to one of the addresses defined for
#                      this service to get ATM address information for
#                      any IP addresses it does not already know the
#                      ATM address of.
#
#   type=bcast   Defines a "broadcast service".  The "addr" field(s)
#                      contain IP addresses of hosts on a given logical
#                      logical ATM network to which copies of any broadcast
#                      packets will be sent, allowing the GigaRouter, when
#                      so configured, to simulate broadcast over a logical
#                      IP network.
#

#Service name=arp0 type=arp addr=47000580ffe1000000f21513eb0020481513eb00

#Service name=bc0 type=bcast addr=198.174.20.1 addr=198.174.22.1 \
#              addr=198.174.21.1

#
# Traffic shaping parameters
#
# Lines beginning with the keyword "Traffic_Shape" define
# traffic shapes which may be used to configure the performance
# characteristics of ATM Virtual Circuits.
#
# The Traffic_Shape's defined here are to be referenced by name when
# to assign traffic shapes to PVC's or Interfaces later in this
# configuration file.  (See Examples in the PVC or Interface section
# of this file for examples on how to reference traffic shapes defined here.)
#
# Each Traffic_Shape entry the ATM configuration file has the following format:
#
# Traffic_Shape name=value peak=bps [sustain=bps burst=cells] [qos=high|low]
#
# The "name" field is a unique name to identify this ATM service, so we
# can refer to the collection of peak, [sustain, burst], [qos] parameters
# as a group when configuring PVC's or Interfaces later in this file.
#
# The 'peak', 'sustain', and 'burst' fields specify, respectively,
# the peak cell rate, the sustained cell rate, and the burst rate.
# The values for 'peak' and 'sustain' are in kilobits per second (maximum
# of 155000), and the value for 'burst' is in cells (maximum of 2048).
#
# The 'qos' (Quality of Service) field specifies which rate queues
# to use.  A value of 'high' corresponds to high priority service
# which uses the high-priority rate queues, and a value of 'low' corresponds
# to low priority service which uses the low-priority rate queues.
#
# The peak rate is the only parameter which is mandatory.  If ommitted,
# the sustain and burst rates are set to match the peak rate.  If qos
# is not specified, it defaults to "high".


#Traffic_Shape name=high_speed_high_quality \
#      peak=155000 sustain=155000 burst=2048 qos=high

#Traffic_Shape name=medium_speed_low_quality \
#      peak=75000 qos=low

#Traffic_Shape name=low_speed_high_quality \
#      peak=15000 qos=high
```

gratm.conf Page 3 of 5

```
# Signalling parameters
#
# Lines beginning with the keyword "Signalling" define
# the signalling protocol which will be used on a physical
# ATM interface to establish Switched Virtual Circuits for
# any logical interfaces on the named physical interface.
#
# Physical interfaces on GigaRouter ATM cards are identified by
# the slot number of the interface card in the GigaRouter chassis
# in hex notation (0-f) plus the location of the physical interface
# on the card (either the top connector, or the bottom connector on the card).
#
# Each Signalling entry the ATM configuration file has the following format:
#
# Signalling card=hex connector=top|bottom [protocol=UNI3.0|UNI3.1|NONE] \
#      [mode=SDH|SONET]  [clock=Ext|Int]
#
# The 'card' and 'connector' specification are mandatory.
#
# The card should be identified by a hexidecimal digit representing
# the slot number of the card in the GigaRouter chassis.
#
# The connector should be either 'top' or 'bottom'.
#
# The 'protocol' parameter defines the signalling protocol to be used
# in the setup of Switched Virtual Circuits (SVC's) on this physical
# interface.  This parameter is optional.  If left unspecified, the
# ATM card uses UNI3.0 signalling by default.
#
# Valid values for the signalling parameter include:
#
#          UNI3.0    - for the UNI 3.0 signalling protocol.
#
#          UNI3.1    - for the UNI 3.1 signalling protocol.
#
#          NONE      - for no signalling protocol.
#
# The 'mode' specification is optional. It can be either SDH or SONET.
# By default, it uses SONET.
#
# The 'clock' specification is also optional.  It can be either Ext(ernal)
# or Int(ernal).  The default setting is Internal clocking.
#
#Signalling card=9 connector=top protocol=UNI3.1
#Signalling card=9 connector=bottom protocol=NONE

#Signalling card=a connector=top protocol=UNI3.1
#Signalling card=a connector=bottom protocol=NONE
#
# Interfaces
#
# Lines beginning with the keyword "Interface" define
# GigaRouter logical ATM interfaces.
#
# The format of a logical interface definition is:
#
# Interface ifname [service=service_name] [traffic_shape=shape_name]
#          [bridge_method=method[,restriction]]
#
# The optional 'service' parameter allows an ATM service to be
# be defined for this logical interface (the 'service_name' must be
# a name defined in the Services section above).
#
# The optional 'traffic_shape' parameter allows a traffic shape
# to be defined for this logical interface (the 'shape_name' must be
# a named defined as a Traffic_Shape above).
#
# If no traffic shape is specified, a default shape of 155Kbps, high
# quality of service is used.
```

gratm.conf Page 4 of 5

```
# The optional 'bridge_method' parameter allows an interface to be used
# for RFC 1483 bridging.  The valid values for the bridge_method parameter are:
#
#     llc_encapsulated          - Use a single PVC, with each frame
#                                 encapsulated with an LLC header to
#                                 identify the protocol
#
#     vc_multiplexed            - Use a separate PVC for each protocol
#
# LLC encapsulated bridging allows any LAN frame type to be transmitted, and
# also allows IP datagrams to be sent directly on the VC.  The optional
# 'restriction' parameter can limit how IP datagrams are routed to the
# interface, and on what kind of LAN frames are transmitted on it.
# The valid values for the restrction parameter are:
#
#     broute_to_ether           - Transmit all routed IP datagrams as
#                                 Ethernet frames
#
#     ether_only                - Transmit all frames (routed IP datagrams and
#                                 all bridged LAN frames) as Ethernet frames
#
#     broute_to_fddi            - Transmit all routed IP datagrams as
#                                 FDDI frames
#
#     fddi_only                 - Transmit all frames (routed IP datagrams and
#                                 all bridged LAN frames) as FDDI frames
#
# Note that unless a restriction or an ARP service is specified, an
# LLC-encapsulated bridging interface will only be able to route to the host at
# the other end of the ATM PVC.

#Interface ga090  service=arp0 traffic_shape=high_speed_high_quality
#Interface ga0980 service=net20 traffic_shape=low_speed_high_quality

#Interface ga0a0  service=arp0 traffic_shape=high_speed_high_quality
#Interface ga0a80 service=net20 traffic_shape=low_speed_high_quality

#Interface ga091  traffic_shape=high_speed_high_quality \
#             bridge_method=llc_encapsulated,broute_to_ether
#
# PVC's
#
# Lines beginning with the keyword "PVC" define
# Permanent virtual circuits.
#
# The format of a PVC definition is:
#
# PVC ifname VPI/VCI \
#             proto=ip|raw|vc|ipnllc|isis|llc[,bridging]|vcmux_bridge,bpro|vc_atmp \
#             [input_aal=3|5|NONE] [traffic_shape=shape] \
#             [dest_if=logical_if [dest_vc=VPI/VCI]]
#
# The first three parameters (ifname, VPI/VCI, and proto) are mandatory.
#
# 'ifname' specifies the GigaRouter ATM logical interface in the usual
# format (e.g., ga030, ga0e80).
#
# 'VPI/VCI' specifies the (decimal) Virtual Path Identifier and
# Virtual Circuit Identifier of the PVC, separated by a slash (/).
```

gratm.conf Page 5 of 5

```
# 'proto' specifies the protocol to be supported on this PVC.  Legal
# values are:
#      'ip'              Internet Protocol (with LLC/SNAP headers)
#      'raw'             raw adaptation layer (AAL-5 or AAL-3/4) packets
#      'isis'            IS-IS packets
#      'llc'             any LLC-encapsulated protocol supported by the GRF
#                         (except for RFC 1483 bridging)
#      'llc,bridging'    any LLC-encapsulated protocol, including RFC 1483
#                         bridging [This is the PVC type for an interface using
#                         bridge_method=llc_encapsulated.]
#      'vcmux_bridge'    bridged packets [This is a PVC type for an interface
#                         using bridge_method=vc_multiplexed.] An additional
#                         parameter specifies the protocol carried on the VC:
#                        ether_fcs          Ethernet frames, with Frame Check Sequence
#                        ether_nofcs        Ethernet frames, without Frame Check Sequence
#                        fddi_fcs           FDDI frames, with Frame Check Sequence
#                        fddi_nofcs         FDDI frames, without Frame Check Sequence
#                        bpdu               802.1D Bridging Protocol Data Units
#      'vc'              IP datagrams [This is a PVC type for an interface using
#                         bridge_method=vc_multiplexed.]
#      'vc_atmp'         atmp home network connections using VC Based
#                         Multiplexing.
#
# If the 'proto' specified is 'raw', the 'dest_if' parameter specifies the
# GigaRouter interface of the destination for this raw adaptation layer
# connection (specified in the same GigaRouter interface format
# described above), and (optionally) the dest_vc parameter specifies
# the destination VPI/VCI.
#
# the 'input_aal' parameter may be used to specify the adaptation layer,
#              input_aal=3    specifies AAL-3/4
#              input_aal=5    specifies AAL-5
#
# The optional 'traffic_shape' parameter allows a traffic shape
# to be defined for this logical interface (the 'shape_name' must be
# a named defined as a Traffic_Shape above).
#
# If no traffic shape is specified, a default shape of 155Kbps, high
# quality of service is used.
#

#PVC ga090  0/32 proto=ip traffic_shape=high_speed_high_quality
#PVC ga0980 0/32 proto=ip traffic_shape=high_speed_high_quality

#PVC ga0a0  1/33 proto=raw traffic_shape=high_speed_high_quality \
#              dest_if=ga090 dest_vc=5/50  input_aal=5

#PVC ga0a80 1/33 proto=ip traffic_shape=high_speed_high_quality

#PVC ga0b0  0/40 proto=isis traffic_shape=high_speed_high_quality
#PVC ga0c0  0/41 proto=isis_ip traffic_shape=high_speed_high_quality

#PVC ga030  0/32 proto=llc,bridging

#PVC ga031  0/32 proto=vcmux_bridge,ether_nofcs
#PVC ga031  0/33 proto=vcmux_bridge,fddi_nofcs
#PVC ga031  0/34 proto=vcmux_bridge,bpdu
```

Figure A-7. Template for the gratm.conf file

grfr.conf file

The `grfr.conf` file configures the Frame Relay protocol on HSSI and SONET cards.

Note: The space before or after equal signs in `grfr.conf` can be there or not, it does not matter. This is true for all of the parameters that use the equal sign. That is why `Enabled = Y` works, and so does `CIR=56000`.

The `grfr.conf` file has eight sections:

–	Link section:	set Frame Relay link (physical port) parameters
–	PVC section:	set Frame Relay route circuit parameters
–	PVCS section:	define Frame Relay switch circuits parameters
–	PVCM1 section:	define Frame Relay 1-way multicast group circuits
–	PVCM2 section:	define Frame Relay 2-way multicast group circuits
–	PVCMN section:	define Frame Relay N-way multicast group circuits
–	PVCEP section:	define an individual endpoint's parameters
–	PVCATMP section:	define GRF home agent connection to home network

grfr.conf Page 1 of 8

```
############################################################################
#   grfr.conf: Frame Relay Configuration
############################################################################
#
#   Link Section: Set Frame Relay Link (physical port) Parameters.
#
#     The first two parameters are mandatory (Slot and Port). Parameters
#     with * prefixed are newly defined parameters.
#
#     Optional Parameters:
#
#      Name:              String: Link Description.  Deafult = "".
#      Enabled:           Y|N: Enable/Disable Link.  Default = Y.
#      LMIType:           None|AnnexA|AnnexD.
#                         Default = None
#
#      N391:  1..255: Polling Intervals per Full Status Message.
#                           Default = 6.
#      N392:  1..10: Error Reporting Threshold.  Default = 3.
#      N393:              1..10: Measurement Interval for mN2.  Default = 4.
#      T391:              5|10|20|25|30: Heartbeat Poll Interval.  Default = 10.
#   *  T392:              5|10|15||20|25|30: Poll Verification Timer. Default = 15.
#   *  Linktype:          UNI-DTE|UNI-DCE|NNI. Default is UNI-DTE.
#      AutoAddGrif:       gs0xx: | Auto | not specified
#                         gs0xx: If LMI reports a new PVC, they will be added
#                         automatically to this interface, using default
#                         values, and Inverse Arp.
#
```

grfr.conf Page 2 of 8

```
#                         Auto: If LMI reports a new PVC the system will attempt
#                         to determine which interface the newly added circuit
#                         belongs and attach the circuit to that interface.
#                         If the parameter is not specified, such PVCs will not
#                         be added (the default behavior).
#               NOTES:
#                         Only Routed and ATMP circuits are affected by this
#                         parameter, switch and multicast circuits are not.
#
#
#    Slot Port Optional Parameters
#    ==== ==== ==================
#link 6    0    name ="Upper Port 6" LMIType = AnnexD  AutoAddGrif = gs060
#link 6    1    LMIType = AnnexD

############################################################################
#
#   PVC Section: Set Frame Relay Route Circuit Parameters.
#
#   Keyword: pvc or pvcr. pvc for backward compatibility
#
#     The first three parameters (Logical Interface, DLCI, and Peer IP
#     address) are mandatory. Parameters with * prefixed are new defined
#     parameters.
#
#      lif:        gsXXX:             One of the logical Interfaces defined in
#                                     grifconfig.conf
#      DLCI:       16..991:           Channel ID.
#      IP Address: 255.255.255.255:   IP Address of station at other end of
#                                     this PVC.  Use 0.0.0.0 to resolve the
#                                     address using Inverse Arp.
#
#     Optional parameters:
#
#      Name:       Quoted String:   PVC name.  Default = "".
#      Enabled:    Y|N:             Enable/Disable PVC.  Default = Y.
#     *CIR:       int:             Committed Information Rate. Default = 55M (bits/sec)
#     *Bc:         int:             Committed Burst Size. Default = 55M (bits/sec)
#     *Be:         int:             Excess Burst Size. Default = 0
#      isis:       Y|N              Y to support ISIS traffic. Default is N.
#
#     Notes: Together, CIR, Be, and Bc are referred as "Traffic Enforcing
#            parameters". Values are in bits.
#
#
#    lif    DLCI Peer IP Address Optional Parameters
#    ===    ==== =============== ==================
#pvc  gs060  405  0.0.0.0         Name="Router1" isis=Y
#pvcr gs060  600  192.0.2.6       Enabled = N  Name="Rotuer2"
#pvc  gs060  505  192.0.2.50      Enabled = Y CIR=56000 Bc=56000 Be=2400
#
############################################################################
#
```

grfr.conf Page 3 of 8

```
################################################################################
#
#   PVCS Section: To define Frame Relay Switch Circuit Parameters.
#
#   Keyword: pvcs
#
#     The first two parameters (Circuit Endpoint-A and Endpoint-B)
#     are mandatory.
#
#      EPA:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#      EPB:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#
#     Optional parameters:
#
#      Name:       Quoted String:    PVC name.  Default = ASCII string
#                                    of EPA. Example: EPA = 1:1:459 then
#                                    name = "1:1:459". Name could be used
#                                    to retrieve status of the circuit.
#     Enabled:    Y|N:              Enable/Disable switch PVC.  Default = Y.
#     CIR:        int:              Committed Information Rate. Default = 55M (bits/sec)
#     Bc:         int:              Committed Burst Size. Default = 55M (bits/sec)
#     Be:         int:              Excess Burst Size. Default = 0
#
#      Notes: Both endpoints, A and B, will have the same CIR, Bc, and
#             Be. These parameters could be overridden using pcvep keyword.
#
#
#     EPA         EPB           Optional Parameters
#     ===         ====          ===================
#pvcs  0:1:199     12:0:98       Name="la_mpls" CIR=64000 Bc=64000 Be=2400
```

grfr.conf Page 4 of 8

```
 ###############################################################################
 #
 #   PVCM1 Section: To define Frame Relay 1-way Multicast Group Circuits.
 #
 #   Keyword: pvcm1
 #
 #     The first (Root Circuit Endpoint), and the next N (leaf Endpoints)
 #     parameters are mandatory. Where 1 <= N <= X. X is limited by the
 #     support hardware.
 #
 #     Theoretically, X could be infinite, but as the value of X gets
 #     higher, the line gets longer, less manageable. Using multiple
 #     pvcm1 lines is an alternative to define a 1-way circuit with
 #     many leaf nodes.
 #
 #      EPR:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
 #      EP1:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
 #      ....         ....             .......
 #      EPN:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
 #
 #     Optional parameters:
 #
 #      Name:       Quoted String:    Group Name.  Default = ASCII string
 #                                    of EPR. Example: EPR = 1:1:459 then
 #                                    name = "1:1:459". Name could be used to
 #                                    retrieve status of the multicast group.
 #     Enabled:    Y|N:              Enable/Disable Multicast group. Default = Y.
 #     CIR:        int:             Committed Information Rate. Default = 55M (bits/sec)
 #     Bc:         int:             Committed Burst Size. Default = 55M (bits/sec)
 #     Be:         int:             Excess Burst Size. Default = 0
 #
 #      Notes: All the leaf endpoints could have been defined as part of
 #             switch circuits with pcvs keyword. The optional parameters
 #             specified here only apply to the root endpoint.
 #
 #             Multiple pcvm1 keywords/lines can be used to define a
 #             multicast group. The parameters of the first line will be used
 #             and the root endpoint must be the same. In the following
 #             example, the first 3 lines define a multicast group of
 #             1 root endpoint and 6 leave endpoints with the name of the
 #             group is "netstar_g".
 #
 #
 #     EPR       EP1       EP2       Optional Parameters
 #     ===       ====      ===       ===================
 #pvcm1 1:0:200   12:1:66   12:1:67   Name="netstar_g"
 #pvcm1 1:0:200   12:1:68   12:1:69
 #pvcm1 1:0:200   12:1:70   12:1:71

 #pvcm1 0:1:199   12:0:98   11:1:49   Name="ascend_g" CIR=64000 Bc=64000 Be=2400
```

grfr.conf Page 5 of 8

```
################################################################################
#
#   PVCM2 Section: To define Frame Relay 2-way Multicast Group Circuits.
#
#   Keyword: pvcm2
#
#     The first (Root Circuit Endpoint), and the next N (leave Endpoints)
#     parameters are mandatory. Where 1 <= N <= X. X is limited by the
#     support hardware.
#
#      EPR:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#      EP1:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#      ....         ....             .......
#      EPN:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#
#     Optional parameters:
#
#      Name:       Quoted String:    Group Name.  Default = ASCII string
#                                    of EPR. Example: EPR = 1:1:459 then
#                                    name = "1:1:459". Name could be used to
#                                    retrieve status of the multicast group.
#     Enabled:    Y|N:              Enable/Disable Multicast group. Default = Y.
#     CIR:        int:              Committed Information Rate. Default = 55M (bits/sec)
#     Bc:         int:              Committed Burst Size. Default = 55M (bits/sec)
#     Be:         int:              Excess Burst Size. Default = 0
#
#             Multiple pcvm2 keywords/lines can be used to define a
#             2-way multicast group. The parameters of the first line
#             will be used and the root endpoint must be the same.
#
#
#     EPR       EP1       EP2       Optional Parameters
#     ===       ====      ===       ==================
#pvcm2 1:0:200   12:1:66   12:1:67   Name="ascend_g"
#pvcm2 1:0:201   12:1:68   12:1:69
#pvcm2 1:0:202   12:1:70   12:1:71

#pvcm2 0:1:199   12:0:98   11:1:49   Name="minn_g" CIR=64000 Bc=64000 Be=2400
```

grfr.conf Page 6 of 8

```
##############################################################################
#
#   PVCMN Section: To define Frame Relay N-way Multicast Group Circuits.
#
#   Keyword: pvcmn
#
#     The first N (Endpoints) parameters are mandatory. Where
#           1 <= N <= X. X is limited by the support hardware.
#
#      EP1:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#      ....         ....             .......
#      EPN:         Slot:Port:DLCI   Slot: 0..15, Port: 0..1, DLCI:16..991
#
#     Optional parameters:
#
#      Name:       Quoted String:    Group Name.  Default = ASCII string
#                                    of EP1. Example: EP1 = 1:1:459 then
#                                    name = "1:1:459". Name could be used to
#                                    retrieve status of the multicast group.
#     Enabled:    Y|N:              Enable/Disable Multicast group. Default = Y.
#     CIR:        int:              Committed Information Rate. Default = 55M (bits/sec)
#     Bc:         int:              Committed Burst Size. Default = 55M (bits/sec)
#     Be:         int:              Excess Burst Size. Default = 0
#
#      Notes: The parameters specified here only apply to all endpoints.
#
#             Multiple pcvmn keywords/lines can be used to define a
#             n-way multicast group the parameters of the first line
#             will be used and the group name must be the same.
#
#
#     EP1       EP2       EP3       Optional Parameters
#     ===       ====      ===       ==================
#pvcmn 1:0:200   12:1:66   12:1:67   Name="ascend_1"
#pvcmn 1:0:200   12:1:65   12:1:64
#pvcmn 1:0:200   12:1:63   12:1:69

#pvcmn 1:0:201   12:1:68   12:1:69   Name="ascend_2"
#pvcmn 1:0:202   12:1:70   12:1:71   Name="ascend_3"

#pvcmn 0:1:199   12:0:98   11:1:49   Name="minn_g" CIR=64000 Bc=64000 Be=2400
```

grfr.conf Page 7 of 8

```
    ############################################################################
    #
    #   PVCATMP Section: Set Frame Relay ATMP Circuit Parameters.
    #
    #   Keyword: pvcatmp  (atmp = Ascend Tunnel M... Protocols)
    #
    #     The first three parameters (Logical Interface, DLCI, and Peer IP
    #     address) are mandatory. Parameters with * prefixed are new defined
    #     parameters.
    #
    #      lif:        gsxxx:            One of the logical Interfaces defined in
    #                                    grifconfig.conf.  Do not set addresses on
    #                                    pvcatmp interfaces in grifconfig.conf.
    #                                    Use the form:    "gsXXX  -  -  -  up"
    #      DLCI:       16..991:          Channel ID.
    #      IP Address: 255.255.255.255:  IP Address of station at other end of
    #                                    this PVC.  Use 0.0.0.0 to resolve the
    #                                    address using Inverse Arp.
    #
    #     Optional parameters:
    #
    #      Name:       Quoted String:    PVC name.  Default = ASCII string
    #                                    of EPA. Example: EPA = 1:1:459 then
    #                                    name = 1:1:459". Name could be used
    #                                    to retrieve status of the circuit.
    #     Enabled:    Y|N:              Enable/Disable switch PVC.  Default = Y.
    #     CIR:        int:              Committed Information Rate. Default = 55M (bits/sec)
    #     Bc:         int:              Committed Burst Size. Default = 55M (bits/sec)
    #     Be:         int:              Excess Burst Size. Default = 0
    #
    #        lif      DLCI Peer IP Address Optional Parameters
    #        ===      ==== ============== =================
#pvcatmp  gs060    505  192.0.2.50      Enabled = Y CIR=56000 Bc=56000 Be=2400
```

grfr.conf Page 8 of 8

```
##############################################################################
#
#   PVCEP Section: To define an individual Endpoint's Parameters.
#
#   Keyword: pvcep
#
#     All parameters are mandatory.
#
#     EP:         Slot:Port:DLCI    Slot: 0..15, Port: 0..1, DLCI:16..991
#     CIR:        int:             Committed Information Rate. Default = 55M (bits/sec)
#     Bc:         int:             Committed Burst Size. Default = 55M (bits/sec)
#     Be:         int:             Excess Burst Size. Default = 0
#
#      Notes: Parameters defined with pvcep keyword override parameters
#             defined by any other keywords. This is designed to provide
#             flexibility in configuring switch/multicast circuits and
#             Asymmetric PVCs.
#
#             For example, to configure an asymmetry circuit, use pvcep
#             to set a different value of one endpoint of the circuit
#             traffic shaping parameters.
#
#     EP         CIR       Bc       Be
#     ===        ====      ==       ==
#pvcep 1:0:200   64000    128000   9600

#######################   END OF GRFR.CONF  ################################
```

Figure A-8. Template for the grfr.conf file

grifconfig.conf file

The `/etc/grifconfig.conf` file defines the interface name, IP address, and netmask for each GRF logical interface. At minimum, the interface name must be established for every configured interface.

When a media card comes on-line, the **`grifconfig`** script consults the contents of the `/etc/grifconfig.conf` file to issue the proper commands that will configure the logical interfaces.

Here is the file format and sample entries for each type of media:

```
# name  address       netmask          broad_dest   arguments
#
gh010   192.0.2.1     255.255.255.0
gf023   192.0.99.1    255.255.255.0    192.0.99.255
ga0364  192.0.130.1   255.255.255.0    192.0.130.10
ga0382  192.0.131.1   255.255.255.0
```

In the example above, the first entry (`gh010`) is the HIPPI media card in slot 1, while the second entry (`gf093`) is logical interface 3 on the FDDI card in slot 2. The third entry refers to the ATM card in slot 3, and specifies a logical interface assigned as permanent virtual circuit (PVC) 64. ATM interface `ga0364` is an explicit point-to-point connection and includes the destination IP address of its point-to-point link in the destination address field.

Name

The `name` entry is the GRF interface name. The GRF interface name has five components that describe an individual interface in terms of its physical slot location in the chassis, and its specific and virtual locations on a media card.

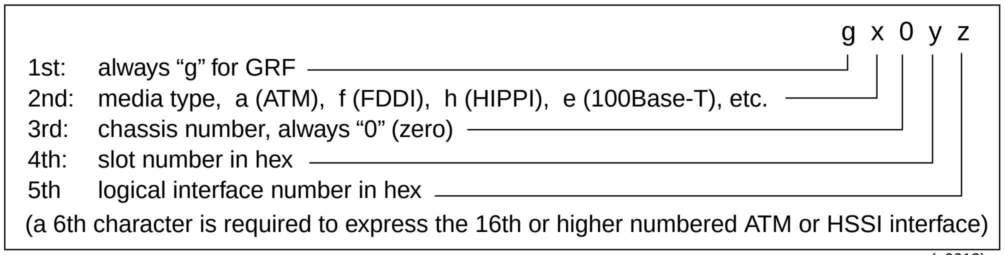

g x 0 y z
1st: always "g" for GRF
2nd: media type, a (ATM), f (FDDI), h (HIPPI), e (100Base-T), etc.
3rd: chassis number, always "0" (zero)
4th: slot number in hex
5th logical interface number in hex
(a 6th character is required to express the 16th or higher numbered ATM or HSSI interface)
(g0012)

Figure A-9. Components in the GRF interface name

Here are examples of interface names:

HIPPI interface names:	`gh030` (single interface per card, only the slot # changes)
ATM interface names:	`ga037f, ga0281, ga01ff`
FDDI interface names:	`gf030, gf021, gf012, gf003`
HSSI interface names:	`gs030, gs021, gs0180`
100Base-T interface names:	`ge030, ge026, ge017`
SONET interface names:	`go030` (single interface per card, only the slot # changes)
T1 interface names:	`gt030`, `gt031`, `gt035`, `gt036`, `gt037` (one per card)

Internet address

The Internet address is the 32-bit IP address for the specified logical interface. The address is in standard dotted-decimal (octet) notation.

Netmask

Netmask is the 32-bit address for the logical IP network on the physical network to which the specific GRF physical interface is attached. The netmask is entered in standard dotted-decimal (octet) notation. If no destination/broadcast address is supplied, a netmask is required.

Broadcast or destination address

The broadcast or destination address is the 32-bit address for this network. Enter the broadcast or destination address in standard dotted-decimal (octet) notation. When a broadcast IP address is assigned to a logical interface, the netmask value is ignored. A dash (-) can be entered in the netmask column, or it can be left blank.

When you assign an IP address to a logical interface on a point-to-point media such as HIPPI or ATM, the destination address is entered in the broadcast address field. Note that any entry in the broadcast address field for HIPPI or ATM makes it a point-to-point connection to that address. If you remove the broadcast address, you create a non-broadcast, multi access interface.

Arguments

In the arguments field, you can specify such items for a logical interface such as its MTU size.

Each interface to run IS-IS must be specifically identified in the `etc/grifconfig.conf` file with an ISO address. An IS-IS interface entry requires `- iso` in the arguments field.

Assigning an ISO address to an interface

To configure an interface to run IS-IS, you create a second entry in `grifconfig.conf` that specifies an ISO address. This entry is in addition to the interface's initial IP address entry.

Here is the ISO address entry and format:

```
# <interface-name> <iso-address> <iso-area> - iso
ga030 49.0000.80.3260.3260.3260.00 49.0000.80 - iso
```

Note that the `iso-area` is the ISO netmask, the dash (-) is a placeholder for destination address, and `iso` is in the arguments field. An example is this set of consecutive entries for ATM interface `go030` in `grifconfig.conf` :

```
ga030    192.0.2.1    255.255.255.0    192.0.2.255
ga030 49.0000.80.3260.3260.3260.00 49.0000.80 - iso
```

This is the `grifconfig.conf` file template:

grifconfig.conf Page 1 of 2

```
#           NetStar $Id: grifconfig.conf,v 1.10.2.3 1997/08/01 17:24:04 pargal Exp $
#
# Configuration file for GigaRouter/GRF interfaces.
#
# The contents of this file specify the IP addressing information for
# the networks attached to the system's interfaces.  This includes
# interfaces on media cards as well as directly attached interfaces
# such as de0 or ef0 (maintenance Ethernet) or lo0 (software loopback).
#
# The addresses of directly attached interfaces are configured
# directly from this file by the /etc/netstart calling the grifconfig(8)
# script.
#
# The addresses of the interface(s) on a given media card are
# configured into the BSD/OS kernel when the media card boots and
# comes on line.
#
# Each entry in this file has the following format:
#
# name        address              netmask         broad_dest      arguments
#
# The name of a GigaRouter interface encodes the hardware type,
# GigaRouter cage number, slot number, and interface number.
#
#   --        The first character must be 'g' (to specify a GigaRouter
#             interface).
#   --        The second character is the hardware type of the
#             interface:  'a' for ATM, 'e' for ETHERNET,
#             'f' for FDDI, 'h' for HIPPI, 'p' for PPP, 's' for HSSI.
#             ('l' is also used, for GigaRouter software loopback.)
#   --        The third character is the number of the GigaRouter cage.
#             (Currently this must be '0', as multiple GigaRouter cages
#             are not yet supported.)
#   --        The fourth character is the hex digit (0 through f) of
#             the slot number within the GigaRouter cage.
#   --        The fifth (and sixth) characters specify the number of the
#             LOGICAL interface on the card:
#
#                     For ATM cards, the fifth and sixth characters are
#                     the hex digits of the logical interface.  Logical
#                     interfaces numbered from 0 to 7f are on the top
#                     physical connector on the ATM card, and logical
#                     interfaces numbered 80 to ff are on the bottom
#                     physical connector.  NOTE:  These logical interface
#                     numbers are NOT the same as the VPI/VCI numbers
#                     of a PVC (see /etc/grpvc.conf for that).
#
#                     For FDDI cards, the fifth character will be 0, 1,
#                     2, or 3 to specify the logical interface on the
#                     FDDI card.  NOTE:  The logical interface number
#                     may be different from the physical interface on
#                     the card, depending on the single- or dual-
#                     attachedness of the various interfaces.  Examples:
#                     "gf073" specifies the bottom-most connector
#                     on the FDDI card in slot 7; "gf020" specifies
#                     top-most connector on the FDDI card in slot 2,
#                     or the top TWO connectors on that card if they're
#                     configured dual-attached.
#
#                     For ETHERNET cards, the fifth character will be 0, 1,
#                     2, 3, 4, 5, 6 or 7 to specify the physical interface
#                     on the ETHERNET card.
#                     Examples:
#                     "ge067" specifies the 8th physical connector
#                                    on the ETHERNET card in slot 6.
#                     "ge000" specifies the first (top) connector on the
#                                    ETHERNET card in slot 0.
#                     "ge0f7" specifies the last (bottom) connector on the
#                                    ETHERNET card in slot 15.
```

grifconfig.conf Page 2 of 2

```
#                    For HIPPI cards, which only have one interface,
#                    the fifth character is always 0.  Example:
#                    "gh0f0" specifies the interface for a HIPPI card
#                    in slot 15.
#
# The IP "address", "netmask" (optional), and "broad_dest" (optional)
# address fields must be specified in canonical IP dotted-quad notation.
# An entry of "-" (a single hyphen) may be specified for any of these
# fields as a place-holder.  This may be useful, e.g., if no netmask
# is desired but a broadcast or destination address must be specified
# in the next field.
#
#
# For this release, the "broad_dest" field specifies the broadcast
# IP address for Ethernet & FDDI interfaces, and the destination of
# a point-to-point ATM or HIPPI interface.
#
# The "arguments" field is for any additional arguments to be supplied
# to the underlying ifconfig(8) command that will be executed by
# grifconfig(8).  The most useful purpose would be to specify an
# MTU value for the interface using the "mtu" keyword of ifconfig(8).
# The keyword "iso" can also be specified here which designates the current
# line as an iso address entry.
# See the example entry below, and the man page for ifconfig(8).
#
#
# NOTE: All interface names are case sensitive ! Always use lower case letters
#            when defining interface names.
#
#
# name       address              netmask         broad_dest       arguments
#
#de0         192.0.2.1            255.255.255.0   192.0.2.255      mtu 1024
#lo0         127.0.0.1            255.0.0.0
#gl000       127.0.1.1
#
# configuration for iso addresses
#
#gf0xx  49.0000.80.3260.3260.3260.00 49.0000.80 -iso
```

Figure A-10. Template for the grifconfig.conf file

grlamap.conf file

Configuration file format

The format of `/etc/grlamap.conf` is:

```
# portcard  logical_addr  dest_portcard
```

where:

`portcard`

is a comma-separated list of media card numbers to which the other fields apply. An * *(asterisk)* indicates all media cards.

Ranges can be used. If media cards 5, 6, 7 and 11 get the same mapping, you can specify “5-7,11”.

`logical_address`

is a comma-separated list of logical addresses to map.
An * *(asterisk)* indicates the default mapping, that is, all logical addresses get mapped.

Ranges can also be used. If logical addresses 0xfc0 through 0xfff get mapped to the same set of media cards, you can specify “0xfc0-0xfff”.

`destination_portcard`

is a comma-separated list of destination media card numbers to which the logical address gets mapped.

Only the first four values are used. Any additional values are flagged as invalid, an error message is printed to the `gr.console` log, and **grlamap** exits. When only three values are designated, the first value fills out the fourth position.

Ranges are not permitted.

Examples of /etc/grlamap.conf entries

To change the default mapping of IP routing for all media cards, enter:

```
# portcard     logical_addr     dest_portcard
    *              0xfc0              0x40
```

To set up that all logical addresses map to media card 5 by default, enter:

```
# portcard     logical_addr     dest_portcard
    *              *                  5
```

To set up that all logical addresses received by media cards
0 through 7 will map to media card 5 by default, enter:

```
# portcard     logical_addr     dest_portcard
 0-7               *                  5
```

This example establishes that for media cards 1, 3, 4, and 15,
the logical addresses 17, 42, and 99 will be sent to one of media cards 6, 8, 13, or 14.
It also sets media card 14 to send these same logical addresses to media cards 6, 8, or 13:

```
# portcard        logical_addr        dest_portcard
 1,3,4,15          0x11,42,99           6,8,13,14
 14                0x11,42,99           6,8,13
```

This is the `grlamap.conf` file template:

grlamap.conf Page 1 of 1

```
#       NetStar $Id: grlamap.conf,v 1.1 1995/02/09 20:29:27 scotth Exp $
#
################################################################################
#
#
# There are 3 fields to determine how logical addresses get
# mapped to destination portcards for each hippi portcard.
#
# The first field is a comma separated list which determines the
# portcard on which the other 2 fields will apply.
# Ranges are allowed. i.e., 1,4-9,15 == 1,4,5,6,7,8,9,15
# '*' signifies that all portcards get this mapping.
#
# The second field is a comma separated list of logical addresses being mapped.
# Ranges are allowed.
# values: 0 - 4095 inclusive, '*' signifies the default (i.e., all LAs).
#
# The third field is a comma separated list of destination port cards.
# Only the first 4 values will be taken as valid, any extra values will
# be flagged invalid, a message will be printed to gr.console & grlamap will
# exit.
# If only 3 values are designated, the first value will be repeated as the
# fourth value.
#
#
# No whitespace is allowed in any field.
#
#
# ie.
# 5,6   997,998         1,0x4,8,15
#
# When port cards 5 & 6 receive an IP packet with a logical address of 997 or
# 998, it will then attempt to randomly forward the packet to one of the
# mapped ports 1, 4, 8 or 15.
#
#
################################################################################

#*      *       5               # default mapping for all LAs for all portcards.
#5      *       6               # default mapping for LAs for portcard 5.

################################################################################
# IP mapping.
################################################################################
*       0xfc0   0x40            # default mapping of IP for all portcards.

################################################################################
#5      1-100   9               # default mapping for LAs for portcard 5.
#5      100-200 4               # default mapping for LAs for portcard 5.
```

Figure A-11. Template for the grlamap.conf file

grppp.conf

This is the `grppp.conf` file template:

grppp.conf Page 1 of 1

```
# Netstar $Id: grppp.conf,v 1.4 1997/03/25 16:54:45 suseela Exp $
#
#
# Template grppp.conf file.
#
# This file is used to set the initial configuration of PPP interfaces.
#
# When a media card configured for PPP is reset, grinchd executes grppp
# to process this file.   The following subset of grppp commands may be
# used in the grppp.conf file.  Most of these commands are used to
# overide default values, and should be used with caution.  Refer to
# the grppp man page for a full explanation of these commands.
#
#       interface INTERFACE_NAME
#       enable negotiation trace
#       set maximum configuration request count = INTEGER
#       set maximum failure count = INTEGER
#       set maximum terminate count = INTEGER
#       set restart timer interval = INTEGER
#       enable lcp magic number
#       set lcp keepalive interval = INTEGER
#       set lcp keepalive packet threshold = INTEGER
#       set lcp mru = INTEGER
#       enable lqr
#       set lqr timer interval = INTEGER
#       enable ipcp
#enable osinlcp
#
# The example below shows the most commonly used grppp commands used in
# a grppp.conf file.

#
# Example Gigarouter PPP initial configuration
#
# interface gs0b0    # Card 11, port 0
#    enable negotiation trace# copy negotiaton traces to /var/log/gr.console
#    enable ipcp# allow IP traffic over PPP
#
# interface gs0b1    # Card 11, port 1
#    enable ipcp# allow IP traffic over PPP
#    enable osinlcp# allow osi traffic over PPP
#
```

Figure A-12. Template for the grppp.conf file

Use this file to set up PPP on HSSI and SONET media cards as well as to enable the IS-IS protocol on those cards. Uncomment the line “`enable osinlcp`” to enable a card to receive IS-IS packets.

grroute.conf file

The `/etc/grroute.conf` file is a table of static routes for GRFs that do not use dynamic routing. This file only needs to include routes to remote networks and hosts, and networks and hosts that do not directly connect to a GRF interface. Routes for directly-connected networks and hosts are created from information in the `/etc/grifconfig.conf` file.

Here is the file format and some sample entries:

```
#   destinationnetmaskgateway/next hop
    192.0.2.0255.255.255.0123.45.67.89
    192.0.2.1255.255.255.255123.45.67.89
```

When a media card comes on-line, the **grroute** script reads the contents of `/etc/grroute.conf` to issue the necessary **route** commands that will add the routes having next hops through that media card's interface(s).

grroute.conf Page 1

```
#    NetStar $Id: grroute.conf,v 1.3 1995/03/15 22:09:07 knight Exp $
#
# grroute.conf -- configuration file for GigaRouter static remote routes
#
# This file should only contain routes to remote networks and
# hosts--i.e., networks and hosts not directly attached to a
# GigaRouter interface.  Routes for networks directly attached
# to the GigaRouter are created as part of configuring the
# GigaRouter interfaces; see /etc/grifconfig.conf.
#
# NOTE:  THIS FILE SHOULD NOT BE USED ON SYSTEMS WITH DYNAMIC
# ROUTING (gated).  If you are running gated, then you should specify
# static routes using the "static" statement in /etc/gated.conf.
#
# Whenever a port card boots, comes on line and has its interface(s)
# configured, the routes specified in this file that are for gateways
# on the network(s) directly attached to those interface(s) are
# configured into the BSD/386 kernel.
#
# The format of each line is follows:
#
#   destination      netmask           gateway/next hop
#
# The destination is the IP address of the remote host or network.
# For the default route, specify a destination of "0.0.0.0" or the
# word "default".
#
# A netmask is required for all entries in this configuration file.
#
# The netmask is normally the mask of the remote network:
#
#    192.0.2.0       255.255.255.0     123.45.67.89
#
# For remote host routes, specify a netmask of 255.255.255.255:
#
#    192.0.2.1       255.255.255.255   123.45.67.89
#
# In the case of the default route (0.0.0.0), the netmask is ignored,
# but some value must be present for the file to parse correctly.
#
#
#    destination     netmask           gateway/next hop
#
#    0.0.0.0         0.0.0.0           192.0.2.2
```

Figure A-13. Template for the grroute.conf file

snmpd.conf file

This is the `snmpd.conf` file template:

snmpd.conf Page 1 of 2

```
#
# Default Agent Configuration File
#
#       This file allows MANAGERS to be specified.  This is used to
#       specify which managers will be receiving which traps.
#
#       Also, COMMUNITYs can be specified. This allows that agent to
#       be configured such that it will only except requests from
#       certain managers and with certain community strings.
#
# GRAMMAR:
#
#       INITIAL         <name> <String>
#
#       TRANSPORT       <name>
#                       [SNMP | SMUX]
#                       [OVER [UNIX | UDP | TCP] [SOCKET | TLI]]
#                       [AT <addr>]
#
#       MANAGER         <addr> [ON TRANSPORT <name>]
#                       [SEND [ALL | NO | traplist] TRAPS
#                               [TO PORT <#> ]
#                               [WITH COMMUNITY <name>]]
#
#       COMMUNITY       <name>
#                       ALLOW op [,op]* [OPERATIONS]
#                       [AS <name>]
#                       [USE encrypt ENCRYPTION]
#                       [MEMBERS                <addr> [,<addr>] ]
#
#
#
#       ALLOW <subagentId> [ON <hostSpec>]
#             WITH <passwordSpec> [<entitySpec>] [<timeout>]
#
#       DENY <subagentId> ON <hostSpec> WITH <passwordSpec>
#
#       ENTITY <EntityName> DESCRIPTION <String>
#
#       LOCAL CONTEXT <contextName> [USES] VIEW <viewName>
#               REFERS TO ENTITY <entityName> AS <oid>
#
#       PROXY CONTEXT <oid> [USES] SOURCE PARTY <oid>
#                                DESTINATION PARTY <oid>
#                                [AND] CONTEXT <oid>
#
#       VIEW <viewName> [[INCLUDE | EXCLUDE] SUBTREE <oid> [MASK <bitmask>]]+
#
#       ALLOW op [,op]* [OPERATIONS] <sugar> SOURCE PARTY <partyName>
#                                      DESTINATION PARTY <partyName>
#                                      [AND] CONTEXT <contextName>
#
#       <partyDefinition> ::= [LOCAL] PARTY <name> ON TRANSPORT <transport>
#                             AT <addr> <AuthPriv>
#                             AS <oid>
#
#       <transport> ::= [ snmpUDPDomain | snmpCLNSDomain | snmpCONSDomain |
#                         snmpDDPDomain | snmpIPXDomain  | rfc1157Domain
#
#       <transport> ::=
#       <AuthPriv> ::= <noAuth> <noPriv> |
#                      <md5Auth> <noPriv> |
#                      <md5Auth> <desPriv>
```

snmpd.conf Page 2 of 2

```
#       <noAuth> ::= <sugar> NO AUTHENTICATION
#
#       <sugar> ::= [AND] [[WITH | USING]]
#
#       <noPriv> ::= <sugar> NO ENCRYPTION
#
#       <md5Auth> ::= <sugar> MD5 AUTHENTICATION <key>
#
#       <key> ::= <sugar> <string> AS KEY
#
#       <desPriv> ::= <sugar> DES ENCRYPTION <key>
#
#       <subagentId>  ::=  SUBAGENT <oid> |
#                          SMUX SUBAGENT <oid> |
#                          UNSPECIFIED SUBAGENTS
#
#       <hostSpec> ::= HOST <hostid> |
#                      UNSPECIFIED HOST[S]
#
#       <passwordSpec> ::= PASSWORD <string>
#                          UNSPECIFIED PASSWORDS
#
#       entitySpec ::= AS ENTITY <entityName>
#
#       <timeout> ::= USING <specificTimeout> TIMEOUT
#       <specificTimeout> ::= <number> SECOND[S] |
#                             NO
#
#       addr ::=        <ip-kind> | <rfc1449addr> | <full-ip>
#       ip-kind ::=     <hostid> |
#                       <hostid> <portid> |
#                       <portid>
#
#       hostid ::=      <hostname> | <ip>
#                               where: hostname is defined in /etc/host
#       portid ::=      [PORT | : ] <#>
#
#       full-ip ::=     <ip>:<#>
#       ip ::=          <#>.<#>.<#>.<#>
#       traplist ::=    trap [, trap]*
#       trap ::=        <trap_name>
#
#
#       op ::=          ALL | GET | SET | TRAP
#       encrypt ::=     NO | <name>
#
#       rfc1449addr ::= tcp_ip_addr | osi_addr
#       tcp_ip_addr ::= <ip>/<#>
#       osi_addr ::= <nsap>/<tsel>
#       nsap ::= hexes
#       tsel ::= hexes
#       hexes = hexbyte[: hexbyte]*
#
#

ALLOW           SUBAGENT 1.3.6.1.4.1.1080.1.1.1
                WITH OTHER PASSWORD
                USE 15 SECOND TIMEOUT

COMMUNITY       public
                ALLOW GET OPERATIONS
                USE NO ENCRYPTION
```

Figure A-14. Template for the snmpd.conf file

15-second time-out

The `snmpd.conf` file template includes an `ALLOW` entry (near the end of the file, as shown above). This entry should be left intact. It gives **snmpd** a 15-second time-out for responses coming from **mib2d** and keeps **snmpd** active should **mib2d** hang.

syslog.conf file

Network logging requires setting up a remote host on the administrative LAN to act as a **syslogd** server. This is the GRF `syslog.conf` file template:

syslog.conf Page 1 of 2

```
##  NetStar $Id: syslog.conf,v 1.6.4.3 1997/09/15 14:57:37 pargal Exp $

#
# To enable the logging of system messages on GRF systems, edit the
# entries in the "Log messages to Network" section below.
#
#      - uncomment the lines in the "Log messages to Network" section
#        of this config file (just below these instructions)
#
#      - change "server.domain.com" to the name of a syslog server on
#        your local network to which this router may send log messages
#
#      - add the IP address of the log server and its host name to /etc/hosts
#        on this GRF system.
#
#      - run "grwrite -v" command to save your changes to
#        /etc/syslog.conf and /etc/hosts
#
#      - add the following lines to the /etc/syslog.conf on your log server:
#        (do not include the # from the first column of this file, ie.,
#        add "local5.* /var/log/mib2d.log" not "#local5.* /var/...")
#
#        #
#        # Syslog configuration for syslog server systems
#        # GRF-specific log files (from GRF systems over the network)
#        #
#        local0.info/var/log/gritd.packets
#        local1.info/var/log/gr.console
#        local2.*/var/log/gr.boot
#        local3.*/var/log/grinchd.log
#        local4.*/var/log/gr.conferrs
#        local5.*/var/log/mib2d.log
#        local6.*/var/log/fred.log
#
#      - kill and restart syslogd on your syslog server machine after making
#        sure that the log files added to the config file exist
#        on the log server machine exist (use the "touch" command
#        to create them if they do not exist, then restart syslogd
#        on the syslog server machine).
#
#      - kill and restart syslogd on the GRF router (or reboot the GRF).
#
# To uncomment, remove "#net# from each line in this section
```

syslog.conf Page 2 of 2

```
#
# Log messages to Network
#
#net#*.err;kern.debug;auth.notice;mail.crit            @server.domain.com
#net#*.notice;kern.debug;lpr,auth.info;mail.crit       @server.domain.com
#net#cron.info                                         @server.domain.com
#net#local0.info                                       @server.domain.com
#net#local1.info                                       @server.domain.com
#net#local2.*                                          @server.domain.com
#net#local3.*                                          @server.domain.com
#net#local4.*                                          @server.domain.com
#net#local5.*                                          @server.domain.com
#net#local6.*                                          @server.domain.com


# ---------------------------------------------------------------------------
# GRF users need not read further - the following configuration examples
# are for IRMS systems
# ---------------------------------------------------------------------------

#
# On IRMS systems (which have sufficient disk space) system messages
# may be logged to disk by uncommenting the lines in the "Log messages to disk"
# section below, touching the log file names to make sure they exist and
# then restarting syslogd.
#
# Note: using the "Log messages to Disk" entries to log messages to
# the RAM-based file system on a GRF system is strongly discouraged
# because the log files can easily fill the RAM file system.
#
# To uncomment, remove "#disk# from each line in this section
#

#
# Log messages to Disk
#
#disk#*.err;*.notice;kern.debug;lpr,auth.info;mail.crit     /var/log/messages
#disk#cron.info                                             /var/log/cron
#disk#local0.info                                           /var/log/gritd.packets
#disk#local1.info                                           /var/log/gr.console
#disk#local2.*                                              /var/log/gr.boot
#disk#local3.*                                              /var/log/grinchd.log
#disk#local4.*                                              /var/log/gr.conferrs
#disk#local5.*                                              /var/log/mib2d.log
#disk#local6.*                                              /var/log/fred.log

*.notice;auth.debug      root
*.emerg                  *
```

Figure A-15. Template for the syslog.conf file

Warranty

B

This appendix contains warranty information for the GR-II, GRF 400, and GRF 1600.

Product warranty

1 Ascend Communications, Inc. warrants that the GRF 400, GRF 1600, and GR-II will be free from defects in material and workmanship for a period of twelve (12) months from date of shipment.

2 Ascend Communications, Inc. shall incur no liability under this warranty if

– the allegedly defective goods are not returned prepaid to Ascend Communications, Inc. within thirty (30) days of the discovery of the alleged defect and in accordance with Ascend Communications, Inc.'s repair procedures; or

– Ascend Communications, Inc.'s tests disclose that the alleged defect is not due to defects in material or workmanship.

3 Ascend Communications, Inc.'s liability shall be limited to either repair or replacement of the defective goods, at Ascend Communications, Inc.'s option.

4 Ascend Communications, Inc. MAKES NO EXPRESS OR IMPLIED WARRANTIES REGARDING THE QUALITY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE BEYOND THOSE THAT APPEAR IN THE APPLICABLE Ascend Communications, Inc. USER'S DOCUMENTATION. Ascend Communications, Inc. SHALL NOT BE RESPONSIBLE FOR CONSEQUENTIAL, INCIDENTAL, OR PUNITIVE DAMAGE, INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS OR DAMAGES TO BUSINESS OR BUSINESS RELATIONS. THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES.

Warranty repair

1 During the first three (3) months of ownership, Ascend Communications, Inc. will repair or replace a defective product covered under warranty within twenty-four (24) hours of receipt of the product. During the fourth (4th) through twelfth (12th) months of ownership, Ascend Communications, Inc. will repair or replace a defective product covered under warranty within ten (10) days of receipt of the product. The warranty period for the replaced product shall be ninety (90) days or the remainder of the warranty period of the original unit, whichever is greater. Ascend Communications, Inc. will ship surface freight. Expedited freight is at customer's expense.

2 The customer must return the defective product to Ascend Communications, Inc. within fourteen (14) days after the request for replacement. If the defective product is not returned within this time period, Ascend Communications, Inc. will bill the customer for the product at list price.

Out-of warranty repair

Ascend Communications, Inc. will either repair or, at its option, replace a defective product not covered under warranty within ten (10) working days of its receipt. Repair charges are available from the Repair Facility upon request. The warranty on a serviced product is thirty (30) days measured from date of service. Out-of-warranty repair charges are based upon the prices in effect at the time of return.

Index