The GNU Gatekeeper

Maintainer of this manual: *Jan Willamowius* <`jan@willamowius.de`> Version 3.9, July 2015

This User Manual explains how to compile, install, configure and monitor *the GNU Gatekeeper (GnuGk)*.

Contents

1 Introduction

1.1 About

The GNU Gatekeeper `<http://www.gnugk.org/>` is an open-source project that implements a H.323 gatekeeper. A gatekeeper provides call control services to H.323 endpoints and is an integral part of most telephony or video conferencing installations that are based on the H.323 standard.

According to the H.323 standard, a gatekeeper shall provide the following services:

- Address Translation
- Admissions Control
- Bandwidth Control
- Zone Management
- Call Control Signaling
- Call Authorization
- Bandwidth Management
- Call Management

The GNU Gatekeeper implements all of these functions and a number of additional features. For the protocol handling and encoding and decoding of the messages, it uses the *H323Plus* `<http://sourceforge.net/projects/h323plus>` library (a continuation of the now defunct OpenH323 project).

H.323 is an international standard published by the *ITU* `<http://www.itu.int/>`. It is a communications standard for audio, video, and data over the Internet. See also Paul Jones' *A Primer on the H.323 Series Standard* `<http://www.packetizer.com/voip/h323/papers/primer/>`.

1.2 Copyright

The GNU Gatekeeper is covered by the *GNU General Public License version 2* (GNU GPL v2). In addition, we explicitly grant the right to link this code to the OpenH323/H323Plus and OpenSSL library.

Some protocols implemented in the GNU Gatekeeper are covered by patents (especially the firewall / NAT traversal protocols). To the best of our knowledge, the GNU Gatekeeper Project has a valid license for all its releases, but users making derived versions of this code must ensure that they have a valid license before enabling those features.

Generally speaking, the GNU GPL allows you to copy, distribute, resell or modify the software, but it requires that all derived works must also be published under the GNU GPL. This means that you must publish full source for all extensions to the gatekeeper and for all programs where you incorporate code from the GNU Gatekeeper. See the file COPYING for details.

If that's not what you want, you must interface to the gatekeeper through the status port and communicate with it via TCP. This allows you to integrate basic functionality into the gatekeeper (and provide source for that) but keep other parts of your application private.

1.3 Name

The full name of this project is *GNU Gatekeeper*, but it may also be referred to as *GnuGk*. The use of "GNU" in the name is to emphasize that this is free software. Early versions were called the *OpenH323 Gatekeeper*.

1.4 Download

The newest version is available at *the download page* `<http://www.gnugk.org/h323download.html>`.

The very latest source code is in the CVS at *Sourceforge Web-GUI* `<http://openh323gk.cvs.sourceforge.net/openh323gk/openh323gk/>`. Beware - that's the bleeding edge.

You can also download executables for a number of operating systems from *the download page* `<http://www.gnugk.org/h323download.html>`.

1.5 Mailing Lists

There are two mailing list for the project, one for developers and one for users.

General user questions should be sent to the *users mailing list* `<mailto:Openh323gk-users@sourceforge.net>`. You can find the list archive *here* `<https://sourceforge.net/mailarchive/forum.php?forum_name=openh323gk-users>`. To join this mailing list, click *here* `<https://lists.sourceforge.net/lists/listinfo/openh323gk-users>`.

To report problems or submit bugs/patches, send email to the *developers mailing list* `<mailto:Openh323gk-developer@sourceforge.net>`. The list archive is *here* `<http://sourceforge.net/mailarchive/forum.php?forum_name=openh323gk-developer>`. Please send user questions to the users mailing list and keep this list for development! If you want to contribute to the project, please *join the developers mailing list* `<http://lists.sourceforge.net/lists/listinfo/openh323gk-developer>`.

1.6 Contributors

The project founder and current maintainer is *Jan Willamowius* `<http://www.willamowius.com/gnugk-support.html>` <*jan@willamowius.de*>

Over the years many people have contributed, most notable Simon Horne, Michal Zygmuntowicz and Chih-Wei Huang.

2 Compiling and Installing

2.1 Pre-requisites for Compiling

To build the gatekeeper you need PTLib and H323Plus. Please see *http://www.gnugk.org/compiling-gnugk.html* `<http://www.gnugk.org/compiling-gnugk.html>` for up-to-date information on required library versions.

To successfully compile the GNU Gatekeeper you must first compile the pre-requisites in this order:

1. PTLib
2. H323Plus

On Unix, run `configure` and `make debugnoshared` or `make optnoshared` in the gatekeeper directory to build debug or release version, respectively.

NOTE: You must use either `make debugnoshared` or `make optnoshared` throughout the compile process. For example, if a library is compiled with `make optnoshared` then everything must be compiled the same way.

2.2 Installing on Unix

The first step is to get an executable: You can either download an executable for your flavour of Unix from *gnugk.org* `<http://www.gnugk.org/h323download.html>`, use the executable your distribution provides or compile the GNU Gatekeeper yourself. For simple installations or to try the features of the gatekeeper, using pre-built executables shouldn't pose any issues, but for professional installations it is always recommended that you self-compile GnuGk.

2.2.1 Installing a binary of GnuGk

Copy the executable to the directory you like and create a config file. There are several config examples and auto startup scripts in the `etc/` subdirectory of the source tree. See section 4.2 (Configuration File) for detailed explanations of the parameters.

For example you may copy GnuGk to `/usr/sbin/`, create a config in `/etc/gatekeeper.ini` and start it by

```
/usr/sbin/gnugk -c /etc/gatekeeper.ini -o /var/log/gnugk.log -ttt
```

See section 4.1 (Command Line Options) for details on the command line options.

2.2.2 Compiling the Gatekeeper

NOTE: you must use GCC 3.3.x or later.

You are strongly encouraged to execute `make debugdepend` or `make optdepend` in the gatekeeper directory before starting actual compilation - these commands build appropriate dependency lists, so any CVS updates to the source code will force all affected files to get recompiled and will prevent the resulting binary from being compiled with a mix of old and updated headers.

Type

```
configure --help
```

to see a detailed list of all compile-time options. You can use them to enable or disable features of the gatekeeper. For example, if you do not need RADIUS just type:

```
configure --disable-radius
```

In order to use the gatekeeper under heavy load, enabling the LARGE_FDSET feature (only available on Unix) is recommended (configure –with-large-fdset=4096). Some systems also need to use ulimit in order to allow more than 1024 sockets to be allocated for a single process. Maximum LARGE_FDSET value for voice calls should be calculated based upon predicted maximum sockets usage using the following formula:

```
MAX_NUMBER_OF_CONCURRENT_CALLS * 10 * 120%

Where:
10 = 2 sockets for Q.931 + 2 sockets for H.245 + 6 sockets for RTP
```

So for 100 concurrent voice calls you don't need more than 1024 sockets in the LARGE_FDSET.

As a final step, you must either use `make debugnoshared` or `make optnoshared`, depending on how you compiled the libraries.

2.3 Installing on Windows

The first step is to obtain the executable program; you can either download it from *gnugk.org* `<http://www.gnugk.org/h323download.html>` or compile the GNU Gatekeeper yourself.

There are two versions of the gatekeeper available: A regular program and a service.

2.3.1 Installing as a Program

These are the steps for a manual installation:

Copy `gnugk.exe` to the folder you like and create a config file. There are several config examples in the `etc/` subdirectory of the download archive. See section 4.2 (Configuration File) for detailed explanations.

Then start the gatekeeper manually from the command line ('cmd.exe') or create a batch file to start it.

For example you may copy GnuGk to `C:\GnuGk\`, create a config in `C:\GnuGk\gatekeeper.ini` and start it as

```
C:\GnuGk\gnugk.exe -c C:\GnuGk\gatekeeper.ini -o C:\GnuGk\gnugk.log -ttt
```

See section 4.1 (Command Line Options) for details on the command line options.

Remember to add GnuGk as an exception for the Windows Firewall so it can communicate freely with the network.

2.3.2 Installing as a Service

These are the steps for a manual installation; there may be a binary version of the Gatekeeper-as-service which includes a GUI installer program available in the download location.

First, ensure that you have the service version of GnuGk before you proceed.

Copy `gnugk.exe` to the folder you like and create a config file named `gatekeeper.ini` in the same folder. See section 4.2 (Configuration File) for detailed explanations. When you run GnuGk as a service, no command line options are available.

To register the service, run the following command from the command line ('cmd.exe'):

```
gnugk.exe install
```

Your service is now installed and will be started on the next reboot, or you may start it manually using the Windows Control Panel -> Services function. On Windows Vista and Windows 7, you may have to disable UAC during the service installation.

When running GnuGk as a service, it will always look for a config file named `gatekeeper.ini` in the current directory. Any changes to the trace level and location of the trace file must be made in the config file rather than the command line.

Remember to add GnuGk as an exception for the Windows Firewall so it can communicate freely with the network.

2.3.3 Compiling the Gatekeeper

Once you have compiled the pre-requisites as specified at the beginning of this section and have the appropriate include/library paths configured, open and compile one of the provided solution files (`.sln`) for your version of Microsoft Visual Studio. If you need database support (MySQL, PostgreSQL, ODBC etc.), install/compile appropriate client libraries before you compile GnuGk.

2.4 The addpasswd utility

Status port authentication and many other authentication modules, like SimplePasswordAuth, require encrypted passwords to be stored in the gatekeeper configuration file. The gatekeeper also supports encryption of all passwords in the config. The `addpasswd` utility is required to generate and store these encrypted passwords. This utility is included with the gatekeeper and can be compiled using:

```
$ cd addpasswd
$ make optnoshared
```

The usage is as follows:

```
$ addpasswd CONFIG SECTION KEYNAME PASSWORD
```

Example 1: 'gkadmin' user with 'secret' password has to be added to the [GkStatus::Auth] config section to enable authentication on the status port:

```
$ addpasswd gatekeeper.ini GkStatus::Auth gkadmin secret
```

Example 2: 'joe' user with 'secret' password has to be added to the [Password] config section to enable endpoint authentication:

```
$ addpasswd gatekeeper.ini Password joe secret
```

Example 3: An encrypted shared secret is added to a RadAuth config section:

```
$ addpasswd gatekeeper.ini RadAuth SharedSecret VerySecretPassword
```

IMPORTANT: The `KeyFilled` variable defines a default initializer for password encryption keys. It can be omitted in the config (and therefore defaults to 0), but if it is specified, each time it changes, encrypted passwords have to be regenerated (encrypted again using the `addpasswd` utility).

3 Getting Started (Tutorial)

3.1 A simple first call

To confirm that all components are up and running, we will use two Linux workstations, both connected to the same LAN. In the examples, the H.323 client is a softphone called "SimpH323" which comes as a sample application with H323Plus in the samples/simple/ folder. If your Linux distribution doesn't include it, you can download the H323Plus source code and compile it yourself or you can use another H.323 endpoint.

On the first server start the *gatekeeper* in direct mode:

```
jan@server1> gnugk -ttt
```

The "`-ttt`" option tells the gatekeeper that it should be verbose and print extra debug output to the console. You can direct the output to a file with "`-o logfilename.log`"

Now, start SimpH323 on another console on the same system:

```
jan@server1> simph323 -l -a -u jan
```

SimpH323 is now listening (`-l`) for calls and will automatically accept them (`-a`). It has also registered with the gatekeeper as user "jan" thereby allowing the gatekeeper to creating an association between the user "jan" and their IP address.

SimpH323 will attempt to automatically locate the gatekeeper, but if the auto detection fails, use "`-g 1.2.3.4`" to specify the IP address.

On the second client run simph323 this way:

```
peter@client2> simph323 -u peter jan
```

This instance of SimpH323 registers with the auto-detected gatekeeper as user "peter" and tries to call user "jan". The gatekeeper will accept the request from "peter" and will determine if it can locate the IP address of a user name "jan".

Because "jan" has already registered with the gatekeeper, it will send "jan"s IP address to "peter". "peter"s SimpH323 will then use that IP address to setup a direct session to "jan"s SimpH323 running on server1.

The instance of SimpH323 on server1 will automatically accept the call and Peter and Jan can chat.

3.2 Using the Status interface to monitor the gatekeeper

The status interface presents a text-based means of interacting with an already-running gatekeeper.

On a new console we use telnet to connect to the gatekeeper:

```
jan@server1> telnet localhost 7000
```

You should receive an "Access forbidden!" message because by default, access to the status port is restricted.

Create a file called `gatekeeper.ini` in the directory where we start the gatekeeper. `gatekeeper.ini` will contain the following three lines:

```
[Gatekeeper::Main]
[GkStatus::Auth]
rule=allow
```

Stop the gatekeeper with Ctrl-C and restart it, but specify that it should use the `gatekeeper.ini` we just created:

```
jan@server1> gnugk -ttt -c ./gatekeeper.ini
```

Use telnet to connect to port 7000 and you should now be allowed to connect to the gatekeeper:

```
jan@server1>  telnet localhost 7000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Version:
Gatekeeper(GNU) Version(2.3.5) Ext(pthreads=1,radius=1,mysql=0,pgsql=0,firebird=0,odbc=0,sqlite=0,
large_fdset=0,crypto/ssl=0,h46018=1,h46023=1,ldap=0,ssh=0) H323Plus(1.22.2) PTLib(2.8.5)
Build(Jul 31 2011, 09:03:11) Sys(Linux x86_64 2.6.32-33-generic)
Startup: Sun, 31 Jul 2011 08:07:36 -0600   Running: 102 days 01:08:15
;
```

Now repeat the first experiment where Peter calls Jan and see which messages are handled by the gatekeeper in non-routed mode.

There are a number of commands that can be issued in the telnet session - type "help" to see them.

To end the telnet session with the gatekeeper type "quit" and hit Enter.

The example configuration file we created is very insecure because it has a default **allow** rule, so there are no restrictions on who can connect to the status port and which commands they may execute.

Change the configuration file to:

```
[Gatekeeper::Main]
[GkStatus::Auth]
rule=password
gkadmin=QC7VyAo5jEw=
```

The fourth line was added by the addpasswd utility, which was used to create a user "gkadmin" with password "secret". This change now enforces authentication to the status port.

Restart the gatekeeper with this new configuration and perform the telnet again. You should now be prompted for a username and password:

```
jan@server1>  telnet localhost 7000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

GnuGk login: gkadmin
Password: secret
Version:
Gatekeeper(GNU) Version(2.3.5) Ext(pthreads=1,radius=1,mysql=0,pgsql=0,firebird=0,odbc=0,sqlite=0,
large_fdset=0,crypto/ssl=0,h46018=1,h46023=1,ldap=0,ssh=0) H323Plus(1.22.2) PTLib(2.8.5)
Build(Jul 31 2011, 09:03:11) Sys(Linux x86_64 2.6.32-33-generic)
Startup: Sun, 31 Jul 2011 08:07:36 -0600   Running: 102 days 01:10:15
;
```

The 4.6 ([GkStatus::Auth]) section contains additional information on securing the status port.

3.3 Running the gatekeeper in routed mode

Starting the gatekeeper in routed mode means that the gatekeeper uses "gatekeeper routed signaling". All signaling messages go through the gatekeeper, giving it much greater control over the calls.

Start GnuGk like this:

```
jan@server2> gnugk -r
```

will put the gatekeeper in routed mode. Telnet to the status port and make a call to see what messages are now handled by the gatekeeper.

Note that all media packets (audio and video) are still sent directly between the endpoints (the 2 instances of SimpH323).

3.4 A virtual PBX: Disconnecting calls

Until now the gatekeeper has acted only as a mechanism to resolve symbolic names to IP addresses. This is a critical function of a gatekeeper, but the gatekeeper is capable of much more.

Because the gatekeeper has a lot of control over the calls, it can also be used to terminate them. While connected to the status port, you can list all active calls with "`PrintCurrentCalls`". To terminate a call, type "`Disconnectip 1.2.3.4`" for one of the endpoints.

For example, a simple script could be written to connect to the status port, obtain a list of ongoing calls and terminate them after 5 minutes to prevent users from using too many system resources.

Other functions such as TransferCall are also available.

3.5 Routing calls to a gateway to reach external users

Without using a gateway you can only call other people with an IP phone over the Internet. To reach people with ordinary telephones you must use a gateway.

```
_________________          ______________
| endpoint "jan"|          |            |
| 192.168.88.35 |--------->| Gatekeeper |
|_______________|          |            |
_________________          |            |
| gateway "gw1" | outgoing |            |
| 192.168.88.37 |<---------|____________|
|_______________|
```

The gatekeeper must be configured to specify which calls should be routed to the gateway and which numbers can be called directly. Use the [RasSrv::GWPrefixes] section of the config file to tell the gatekeeper the prefix of numbers that should be routed to the gateway.

```
[RasSrv::GWPrefixes]
gw1=0
```

This entry tells the gatekeeper to route all calls to E.164 numbers starting with "0" to the gateway that has registered with the H.323 alias "gw1". If there is no registered gateway with that alias the call will fail.

NOTE: You must use the gateway alias - you cannot use the IP address of the gateway.

A prefix can contain digits `0-9`, `#` and `*`. It can also contain a special character `.` (a dot) that matches any digit and can be prefixed with `!` (an exclamation mark) to disable the prefix. Prefix matching is done according to the longest matching prefix rule, with ! rules having higher priority if lengths are equal. You may also use := syntax to set the priority between several gateways matching the same prefix (see section 7.2 ([RasSrv::GWPrefixes]) for details). Some examples:

```
[RasSrv::GWPrefixes]
; This entry will route numbers starting with 0048 (but not with 004850 and 004860)
; to gw1
gw1=0048,!004850,!004860
; This entry will match only 001 with 10 digits following and route the call to
; gw2
gw2=001..........
```

3.6 Rewriting E.164 numbers

When using a gateway you often have to use different numbers internally and rewrite them before sending them over a gateway into the telephone network. You can use the 6.3 ([RasSrv::RewriteE164]) section to configure that.

Example: You want to call number 12345 with your IP Phone and would like to reach number 08765 behind a gateway called "gw1".

```
[RasSrv::GWPrefixes]
gw1=0

[RasSrv::RewriteE164]
12345=08765
```

You can also configure rewriting of E.164 numbers based on which gateway you are receiving a call from or sending a call to using the 6.4 ([RasSrv::GWRewriteE164]) section.

Example: You have two different gateways ("gw1" and "gw2") which you are sending calls with prefix 0044 to, but which require a different prefix to be added to the number after the routing has selected the gateway. This might be for identification purposes for example.

```
[RasSrv::GWPrefixes]
gw1=0044
gw2=0044

[RasSrv::GWRewriteE164]
gw1=out=0044=77770044
gw2=out=0044=88880044
```

Example: You want to identify calls from a particular gateway "gw1" with a specific prefix before passing these calls to another gateway "gw2".

```
[RasSrv::GWPrefixes]
gw2=1

[RasSrv::GWRewriteE164]
gw1=in=00=123400
```

Rewrite expressions accept dot '.' and percent sign '%' wildcard characters to allow building more general rules. The dot character can occur on both the left and right hand sides of expressions. The percent sign can occur only at the left side. Use '.' to match any character and copy it to the rewritten string and '%' to match any character and skip it. A few simple examples:

```
[RasSrv::RewriteE164]
; Rewrite 0044 + min. 7 digits to 44 + min. 7 digits
0044.......=44.......
; Rewrite numbers starting with 11 + 4 digits + 11  to 22 + 4 digits + 22
; (like 11333311 => 22333322, 110000112345 => 220000222345)
11....11=22....22
; strip the first four digits from all numbers (11114858345 => 4858345)
; this is equivalent of 10 rules %%%%1=1, %%%%2=2, ...
%%%%.=.
; insert two zeros in the middle of the number (111148581234 => 11110048581234)
....48=....0048
; even this is possible (415161 => 041051061)
4.5.6=04.05.06
```

3.7 Using IPv6

To use IPv6 with GnuGk, you must enable it in the config file:

```
[Gatekeeper::Main]
EnableIPv6=1
```

Calls between IPv4 and IPv6 endpoints are automatically put into proxy-mode to allow GnuGk to perform address translation. If your endpoints can automatically handle mixed IPv4-IPv6 calls the auto-proxying can be disabled using the `AutoProxyIPv4ToIPv6Calls` switch in the [RoutedMode] section. As of 2011-11-10 there don't appear to be any endpoints which can do this.

To support IPv4 and IPv6 endpoints at the same time, GnuGk relies on the operating system to manage IPv4 mapped IPv6 addresses. With a few exception, most current operating systems support this.

Operating System Overview:

- **Linux** OK
- **Windows 7** OK
- **Windows Server 2008** OK
- **Windows Vista** OK
- **Windows XP** either IPv4 or IPv6
- **FreeBSD** OK
- **NetBSD** OK
- **OpenBSD** either IPv4 or IPv6
- **Solaris** OK

For Windows, you need at least Windows Vista, Windows Server 2008, Windows 7 or newer. On Windows XP GnuGk will run as a IPv6-only gatekeeper if you enable IPv6 support. OpenBSD doesn't support IPv4 mapped addresses at all (latest version tested: OpenBSD 5.0), so it can only run GnuGk as either an IPv4 or IPv6 gatekeeper.

As of December 2011, IPv6 support in endpoints is known to work with the following:

- Most Tandberg devices with a recent firmware support IPv6 (eg. C series, EX90 or VCS).
- Polycom HDX endpoints with firmware 3.0 or higher.
- Spranto 2.6.0.14 or higher.

Known to not work:

- LifeSize endpoints do not support an IPv6 gatekeeper.

3.8 Using servers with multiple IPs

By default GnuGk will listen to all IPs on a server and will automatically select the correct sending IP to reach an endpoint. There are a number of config switches to select which IPs to use specifically.

With Home= you can select the interfaces GnuGk should listen on. Usually you would select 1 or 2 interfaces on a machine with multiple IPs. Thats something every user might consider.

With Bind= you can select which IP to use for outgoing messages. This can be useful if your gatekeeper listens to many IPs, but it can also have some non-obvious consequences and this switch should be avoided by most users.

Another related switch is ExternalIP= which can be used to send different IPs inside of your messages than you are actually listening on. This can be usefull if you are doing port forwarding, but should also be avoided and you should use one of the firewall traversal protocols instead.

3.9 Enabling Audio and Video Encryption

You can configure GnuGk as an encryption proxy to ensure that more or all outgoing calls are encrypted, whether your endpoint support encryption themselves or not.

First, enable "half call media" which means GnuGk will add encryption if only one side of the call supports encryption. This will enable encryption for those of your endpoints that might not support encryption by themselves. You can also set if you want 128 or 256 bit AES. (Check "h235media=1" in the startup message to make sure your GnuGk has the encryption features enabled.)

```
[RoutedMode]
EnableH235HalfCallMedia=1
H235HalfCallMediaStrength=256
```

To make sure no call goes through without encryption, you can set

```
[RoutedMode]
RequireH235HalfCallMedia=1
```

When you have this switch on, calls without encryption will be aborted.

Finally, you can take precautions that its always the "outside" connection that gets encryption added. The GnuGk feature is "half call media" and you have to make sure its not only the internal half of the call that gets encrypted. Thus you can remove the encryption from all endpoint on your internal network and with the above settings GnuGk will add encryption to all outgoing calls.

```
[RoutedMode]
RemoveH235Call=192.168.1.0/24, 10.0.1.0/32
```

The next step after media encryption would be to add 12.11 (TLS (transport layer security)) encryption to the signalling channel.

4 Basic Gatekeeper Configuration

The behavior of the *gatekeeper* is determined by the command line options and configuration file. Some command line options may override a setting from the configuration file. For example, the option `-l` overrides the setting `TimeToLive` in the configuration file.

4.1 Command Line Options

Almost every option has a short and a long format, e.g., `-c` is the same as `-config`.

4.1.1 Basic

`-h -help`

Show all available options and quit the program.

`-c -config filename`

Specify the configuration file to use.

`-strict`

Strict configuration check (don't start with config errors)

`-s -section section`

Specify which main section to use in the configuration file. The default is [Gatekeeper::Main].

`-l -timetolive n`

Specify the time-to-live timer (in seconds) for endpoint registration. Overrides the setting `TimeToLive` in the configuration file. See 4.5 (there) for detailed explanations.

`-b -bandwidth n`

Specify the total bandwidth available for the gatekeeper in units of 100 bits per second. Without this option, bandwidth management is disabled.

`-pid filename`

Specify the pid file. Only valid for Unix version.

`-u -user name`

Run the gatekeeper process as this user. Only valid for Unix version.

`-core n`

Enable writing core dump files when the application crashes. A core dump file will not exceed n bytes in size. A special constant "unlimited" may be used to not enforce any particular limit. Only valid on Linux.

`-mlock`

Lock GnuGk into memory to prevent it being swaped out. Only valid on Linux.

4.1.2 Gatekeeper Mode

The options in this subsection override the settings in the 5.1 ([RoutedMode] section) of the configuration file.

`-d -direct`

Use direct endpoint call signaling.

`-r -routed`

Use gatekeeper routed call signaling.

`-rr -h245routed`

Use gatekeeper routed call signaling and H.245 control channel.

4.1.3 Debug Information

`-o -output filename`

Write trace log to the specified file.

`-t -trace`

Set trace verbosity. Each additional `-t` adds additional verbosity to the output. For example, use `-ttttt` to set the trace level to 5.

4.2 Configuration File

The *GNU Gatekeeper* configuration file is a standard text file. The basic format is:

```
[Section String]
Key Name=Value String
```

Comments are marked with a hash (`#`) or a semicolon (`;`) at the beginning of a line.

The file `complete.ini` contains all available sections for GnuGk. In most cases it doesn't make sense to use them all at once. The file is just meant as a collection of examples for many settings.

The configuration file can be changed at run time. Once you modify the configuration file, you may issue the `reload` command via the status port, or send the `HUP` signal to the gatekeeper process:

```
kill -HUP ‘cat /var/run/gnugk.pid‘
```

4.3 Database Configuration

All GnuGk modules that use a database (eg. 8.8 ([SQLAuth]), 9.5 ([SQLAcct]) etc.) support a common set of configuration parameters that is described here. You have to repeat all settings for each module, even if they are the same. But you are also free to use differend database drivers and options for each module.

- `Driver=MySQL | PostgreSQL | Firebird | ODBC | SQLite`
  Default: `N/A`
  Database driver to use. Currently, `MySQL`, `PostgreSQL`, `Firebird`, `ODBC` and `SQLite` drivers are implemented. The MySQL driver can also be used for MariaDB and other MySQL forks. Not all of these driver are always available. When GnuGk is compiled, only those drivers are included where the

necessary database libraries and header files are available. When you start GnuGk, you can see in the version string which drivers are included in your executable. At runtime, GnuGk will load the shared library (DLL) for the database you have configured.

GnuGk supports version 3 of SQLite.

Make sure your database is configured to support password-based authentication - Microsoft SQL Server must use "Mixed Mode" for this feature to function properly.

- `Library=c:/Program Files/Mysql/libmysql.dll`
  Default: `N/A`
  If the shared library or DLL is not found automatically, you can set a different filename here or provide an absolute path to the library.
- `Host=DNS[:PORT] | IP[:PORT]`
  Default: `localhost`
  SQL server host address. Can be in the form of `DNS[:PORT]` or `IP[:PORT]`. Like `sql.mycompany.com` or `sql.mycompany.com:3306` or `192.168.3.100`. The ODBC driver will ignore this setting.
- `Database=billing`
  Default: `N/A`
  The database name to connect to.

  To connect to an **ODBC data source from a Windows server**, create the data source though Control Panel / Administrative Tools / Data Sources (ODBC) and add a System DSN. Use the name of the System DSN in GnuGk's Database= setting.

  To connect to an **ODBC data source from a Unix server**, use a DSN definition configured in unixODBC. Some unixODBC drivers seem to ignore the Username and Passwort set in the GnuGk config. For those, you should use a DSN of the form `DSN=GnuGk;UID=admin;PWD=secret;`. Even in this case, GnuGk's Usename= and Password= settings must always be present. Depending on your unixODBX configuration, you might have to `export ODBCINI=/etc/unixODBC/odbc.ini` and `export ODBCSYSINI=/etc/unixODBC` before starting GnuGk.
- `Username=gnugk`
  The username used to connect to the database.
- `Password=secret`
  The password used to connect to the database. If the password is not specified, a database connection attempt without any password will be made. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password to connect to the database is in an encrypted form and should be created using the `addpasswd` utility.
- `MinPoolSize=5`
  Default: `1`
  Define the number of active SQL connections. This allows for better performance under heavy load, because more than 1 concurrent query can be executed at the same time. Setting `MinPoolSize=1` will simulate the old behavior, when access to the SQL database was serialized (one query at time). Don't let the name fool you, this is the exact number of connections.

4.3.1 Placeholders in queries

Many SQL modules provide a set of placeholders that you can use in your queries, like %{CallId} in SqlAcct.

Placeholders allways start with the percent sign. Beware that you must escape the percent sign if you need it for something else in your queries (eg. in a LIKE). One way to do so is to use CHAR(37), eg. concat(alias,CHAR(37)) instead of concat(alias,'%').

4.3.2 Stored Procedures

Stored procedures work very well when using MySQL.

When using ODBC, you can not call stored procedures which use parameters using the "CALL Procedure-Name" syntax, but you can call them with "EXEC ProcedureName".

When using stored procedures for accounting, make sure they return at least one dummy row, so GnuGk won't drop the call.

4.4 Regular Expressions

In a few places in the configuration file, GnuGk allows regular expressions. The syntax for these regular expressions is "extended POSIX 1003.2 regular expressions". On Unix systems you can usually get a manual page explaining the syntax with "man 7 regex" or see it online at *http://www.kernel.org/doc/man-pages/online/pages/man7/regex.7.html* `<http://www.kernel.org/doc/man-pages/online/pages/man7/regex.7.html>`.

4.5 Section [Gatekeeper::Main]

- `Name=GnuGk`
  Default: `OpenH323GK`
  Gatekeeper identifier of this gatekeeper. The gatekeeper will only respond to GRQs for this ID and will use it in a number of messages to its endpoints.

- `EnableIPv6=1`
  Default: `0`
  If GnuGk has been compiled with IPv6 support, you can use this switch to turn it on.

- `Home=192.168.1.1`
  Default: `listen to all IPs`
  The gatekeeper will listen for requests on this IP address. If not set, the gatekeeper will listen on all IPs of your host. Multiple Home addresses can be used and must be separated with a semicolon (;) or comma (,).

- `NetworkInterfaces=192.168.1.1/24,10.0.0.1/0`
  Default: `N/A`
  Specify the network interfaces of the gatekeeper. By default the gatekeeper will automatically detect the interfaces of your host, so this setting is not usually required, but is available if automatic detection fails. If you are using GnuGk behind a NAT box then you should use the ExternalIP setting (described below) which will automatically configure GnuGk to operate as if it was on the NAT box. The ExternalIP setting will take precedence and will override this value.

  NOTE: If this setting is changed, you must restart the gatekeeper. A reload from the status port will not cause this value to be re-read.

- `Bind=192.168.1.1`
  Default: `N/A`
  Specify the IP address for default routing. Use this to specify which default IP address to use in a multihomed virtual environment where there may be many virtual interfaces on one host.

- `EndpointIDSuffix=_tgdz646438`
  Default: `_endp`
  The gatekeeper will assign a unique identifier to each registered endpoint. This option can be used to

specify a suffix to append to the endpoint identifier. This option useful for security to make it harder for an attacker to guess endpoint IDs and should be set to a value that can't easily be guessed.

This setting doesn't change when the config is reloaded, you must do a full restart!

- `TimeToLive=300`
  Default: `-1`
  An endpoint's registration with a gatekeeper may have a limited life span. The gatekeeper specifies the registration duration for an endpoint by including a **timeToLive** field in the RCF message. After the specified length of time, the registration is considered expired. The endpoint must periodically send a RRQ having the **keepAlive** bit set prior to the expiration time. Such a message may include a minimum amount of information as described in H.225.0 and is known as a lightweight RRQ.

  The endpoint may request a shorter **timeToLive** in the RRQ message to the gatekeeper.

  To avoid an overload of RRQ messages, the gatekeeper automatically resets this timer to 60 seconds if you specify a lower value.

  After the expiration time, the gatekeeper will make two attempts using IRQ messages to determine if the endpoint is still alive. If the endpoint responds with an IRR, the registration will be extended. If not, the gatekeeper will send a URQ with reason **ttlExpired** to the endpoint. The endpoint must then re-register with the gatekeeper using a full RRQ message.

  To disable this feature, set it to `-1`.

- `EnableTTLRestrictions=0`
  Default: `1`
  The default TimeToLive (TTL) configured via the "TimeToLive" parameter does not apply to endpoints which use the H.460.17 and H.460.18 protocols to traverse a firewall. In order to keep the firewall pinhole open the TimeToLive for those endpoints defaults to 19 seconds. The "TimeToLive" parameter would allow you to change this to as low as 5 seconds, and as high as 30 seconds.

  If you know for sure that there is no need for a keep-alive, you can disable these restrictions by setting this switch to 0. If TTL restrictions are disabled, the TimeToLive becomes a global setting for all endpoints, including H.460.17 and H.460.18.

- `CompareAliasType=0`
  Default: `1`
  By default, a H323ID of '1234' won't match E164 number '1234' when comparing aliases. This parameter allows you to ignore the alias type when performing comparisons.

- `CompareAliasCase=0`
  Default: `1`
  By default, alias 'jan' won't match alias 'Jan'. If set to false, the comparison will not be case sensitive.

- `TraceLevel=2`
  Default: `0`
  Set trace level (same as -t on the command line).

- `TotalBandwidth=100000`
  Default: `-1`
  Total bandwidth available for all endpoints in units of 100 bits per second (eg. 5120 means 512 kbps). By default this feature is off (-1).

  NOTE: At this time, the GnuGk only checks calls to and from registered endpoints.

- `MinimumBandwidthPerCall=1280`
  Default: `-1`
  Raise bandwidth requests from endpoints to at least this value in units of 100 bits per second. The

value includes both directions, so a 384 kbps call would have a value of 7680. Setting a minimum is useful when endpoints don't report correct values (eg. Netmeeting). If set to zero or less, no minimum is enforced (default).

NOTE: At this time, the GnuGk only checks calls to and from registered endpoints.

- `MaximumBandwidthPerCall=100000`
  Default: `-1`
  Set maximum bandwidth allowed for a single call in units of 100 bits per second. If set to zero or less, no maximum is enforced (default).

  NOTE: At this time, the GnuGk only checks calls to and from registered endpoints.

- `RedirectGK=Endpoints > 100 | Calls > 50`
  Default: `N/A`
  This option allow you to redirect endpoints to alternate gatekeepers if the gatekeeper becomes overloaded. In the example above, the gatekeeper will reject a RRQ if the number of registered endpoints would exceed 100, or reject an ARQ if concurrent calls exceed 50.

  Furthermore, you may explicitly redirect all endpoints by setting this option to `temporary` or `permanent`. The gatekeeper will send a RAS rejection message with a list of alternate gatekeepers defined in `AlternateGKs`. Note that a `permanent` redirection means that the redirected endpoints will not register with this gatekeeper again. **NOTE:** The redirect capability will only function with H.323 version 4 compliant endpoints.

- `AlternateGKs=1.2.3.4;1719;false;120;GnuGk`
  Default: `N/A`
  If the endpoint loses connectivity with GnuGk it should automatically try to register with the alternate gatekeeper specified here.

  NOTE: Depending on the endpoint, it may not attempt to re-establish a connection to its original gatekeeper. Support for "Assigned Gatekeepers" was added in H.323v6. See *http://www.packetizer.com/ipmc/h323/whatsnew_v6.html* `<http://www.packetizer.com/ipmc/h323/whatsnew_v6.html>` for additional information.

  The primary gatekeeper includes a field in the RCF to inform endpoints which alternate IP and gatekeeper identifier to use.

  The alternate gatekeeper needs to be aware of all registrations on the primary gatekeeper or else it would reject calls. Our gatekeeper can forward every RRQ to an alternate IP address.

  The AlternateGKs config option specifies the fields contained in the primary gatekeeper's RCF. The first and second fields of this string define where (IP, port) to forward to. The third tells endpoints whether they need to register with the alternate gatekeeper before placing calls. They usually don't because we forward their RRQs, so they are automatically known to the alternate gatekeeper. The fourth field specifies the priority for this gatekeeper. Lower is better; usually the primary gatekeeper is considered to have priority 1. The last field specifies the alternate gatekeeper's identifier.

  You may specify multiple alternate gatekeepers as a comma separated list.

  This global definition can be overriden by a per IP specification in 7.8 ([RasSrv::AlternateGatekeeper]).

  NOTE: In older versions of GnuGk, the fields were separated by colons. Since version 2.3.3 this is deprecated and you must change your configurations to use semicolons.

- `SendTo=1.2.3.4:1719`
  Default: `N/A`
  Although this information is contained in AlternateGKs, you must still specify which address to forward RRQs to. This might differ from AlternateGK's address due to multihomed systems, so it's a separate config option.

You can specify multiple gatekeepers in a comma separated list.

- `SkipForwards=1.2.3.4,5.6.7.8`
  Default: `N/A`
  To avoid circular forwarding, you shouldn't forward RRQs you get from the other gatekeeper (this statement is true for both primary and alternate gatekeeper). Two mechanisms are used to identify whether a request should be forwarded. The first one looks for a flag in the RRQ. Since few endpoints implement this, we can increase the overall reliability of the system by specifying it here.

  Specify the other gatekeeper's IP in this list.

- `StatusPort=7000`
  Default: `7000`
  Status port to monitor the gatekeeper. See 13 (this section) for details.

  The GNU Gatekeeper will listen to the status port on all IPs it is listening for call signaling. You should protect those ports in your firewall and set access control rules in the 4.6 ([GkStatus::Auth]) section for those you can't close completely.

- `StatusTraceLevel=2`
  Default: `2`
  Default output trace level for new status interface clients. See 13 (this section) for details.

- `MaxStatusClients=5`
  Default: `20`
  Specifies the maximum number of concurrent connections on the status port. To disable any connections to the status port, set this switch to 0.

- `SshStatusPort=1`
  Default: `0`
  Use the SSH protocol for the status port. User passwords can be set in the 4.6 ([GkStatus::Auth]) section.

  Example connection command:

  ```
  ssh -p 7000 gnugk@1.2.3.4
  ```

- `StatusEventBacklog=20`
  Default: `0`
  Set the number of status port events that are saved for later display in a ring buffer.

- `StatusEventBacklogRegex=^[RA]RJ`
  Default: `.`
  Define a regular expression to restrict which status port events are saved in the backlog. By default all events are saved.

- `TimestampFormat=ISO8601`
  Default: `Cisco`
  This setting configures the default format of timestamp strings generated by the gatekeeper. This option affects 9.5 ([SQLAcct]), 9.4 ([RadAcct]), 9.3 ([FileAcct]) and other modules, but not 12.1 ([CallTable]). You can further customize timestamp formatting per module by configuring the `TimestampFormat` setting in the module-specific configuration portion of the config file.

  There are four predefined formats:

  - `RFC822` - a default format used by the gatekeeper (example: Wed, 10 Nov 2004 16:02:01 +0100)
  - `ISO8601` - standard ISO format (example: 2004-11-10 T 16:02:01 +0100)

  - `Cisco` - format used by Cisco equipment (example: 16:02:01.534 CET Wed Nov 10 2004)
  - `MySQL` - simple format that MySQL can understand (example: 2004-11-10 16:02:01)

  If none of the predefined options is suitable, you can build your own format string using rules from the `strftime` C function (see man strftime or search MSDN for strftime). In general, the format string consists of regular character and format codes, preceded by a percent sign. Example: "%Y-%m-%d and percent %%" will result in "2004-11-10 and percent %". Some common format codes:

  - `%a` - abbreviated weekday name
  - `%A` - full weekday name
  - `%b` - abbreviated month name
  - `%B` - full month name
  - `%d` - day of month as decimal number
  - `%H` - hour in 24-hour format
  - `%I` - hour in 12-hour format
  - `%m` - month as decimal number
  - `%M` - minute as decimal number
  - `%S` - second as decimal number
  - `%y` - year without century
  - `%Y` - year with century
  - `%u` - microseconds as decimal number (**this is a GnuGk extension**)
  - `%z` - time zone abbreviation (+0100)
  - `%Z` - time zone name
  - `%%` - percent sign

- `EncryptAllPasswords=1`
  Default: `0`
  Enable encryption of all passwords in the config (SQL passwords, RADIUS passwords, [Password] passwords, [GkStatus::Auth] passwords). If enabled, all passwords must be encrypted using the `addpasswd` utility. Otherwise only [Password] and [GkStatus::Auth] passwords are encrypted (old behavior).

- `KeyFilled=0`
  Default: `N/A`
  Define a global padding byte to be used during password encryption/decryption. It can be overridden by setting `KeyFilled` within a particular config section. Usually, you do not need to change this option.

Most users will never need to change any of the following values. They are mainly used for testing or very sophisticated applications.

- `UseBroadcastListener=0`
  Default: `1`
  Defines whether to listen to broadcast RAS requests. This requires binding to all interfaces on a machine, so if you want to run multiple gatekeepers on the same machine you should turn this off.

- `UnicastRasPort=1719`
  Default: `1719`
  The RAS channel TSAP identifier for unicast.

- `UseMulticastListener=0`
  Default: `1`
  Enable or disable gatekeeper discovery using IPv4 multicast. By default it is enabled.

- `MulticastPort=1718`
  Default: `1718`
  The RAS channel TSAP identifier for IPv4 multicast.

- `MulticastGroup=224.0.1.41`
  Default: `224.0.1.41`
  The IPv4 multicast group for the RAS channel.

- `EndpointSignalPort=1720`
  Default: `1720`
  Default port for call signaling channel of endpoints. Used when searching for endpoints in the registration table when the actual call signaling port is unknown.

- `ListenQueueLength=1024`
  Default: `1024`
  Queue length for incoming TCP connection.

- `StatusSendBufferSize=16384`
  Default: `16384`
  Set the TCP send buffer size for status port connections.

- `StatusReceiveBufferSize=16384`
  Default: `16384`
  Set the TCP receive buffer size for status port connections.

- `ExternalIP=myip.no-ip.com`
  Default: `N/A`
  When using GnuGk behind a NAT you can set the external IP address that you wish the gatekeeper to masquerade as. This will allow external endpoints and other gatekeepers to contact the NATed gatekeeper. To work you must enable proxy-mode and port forward the required ports to the gatekeeper IP or put the gatekeeper in the NAT box DMZ. This is different than the bind setting, which specifies a physical IP address on the GnuGk box.

  You may specify an IP address or a fully-qualified domain name (FQDN). If you use a FQDN and `ExternalIsDynamic` is set to false, it will be resolved to an IP address on startup or configuration reload. If `ExternalIsDynamic` is set to true, the name will be stored and resolved when needed.

- `ExternalIsDynamic=1`
  Default: `0`
  Configures the GnuGk to support an external dynamic address. If enabled, GnuGk will ensure that the Dynamic DNS (DDNS) service receives keep-alive messages to maintain your DDNS name lease. You must also configure the `ExternalIP` setting with a DNS address maintained by a DDNS service such as www.dyndns.com or www.no-ip.com.

- `DefaultDomain=gnugk.org,gnugk.de`
  Default: `N/A`
  If the GnuGk receives a request for an address in the format **user@domain.com**, this option will strip the domain from the address if it matches the `DefaultDomain` setting and will then process the request using just the "**user**" field. This is useful when receiving interdomain calls placed via SRV routing policy where the full URI is received. It can also be used in conjunction with the [RasSrv::RewriteAlias] section to convert the received URI into an E.164 number for further processing and routing.

- `Authenticators=H.235.1,CAT`
  Default: `H.235.1,MD5,CAT`
  Selects the specific authenticators to use when authenticating endpoints. The default options are: H.235.1 (HMAC SHA1 / old H235AnnexD), MD5 (Digest Authentication) and CAT (Cisco Access Tokens ie RADIUS). Setting a value of NONE will disable authenticators. The order indicates the priority of the authentication mechanism. If this setting is omitted, all authenticators are loaded by default. If you are using plugin authenticators, then you may want to disable the default authenticators to provide optimum security. Note: H.235.1 requires OpenSSL support compiled into GnuGk.

- `DisconnectCallsOnShutdown=0`
  Default: `1`
  GnuGk will disconnect all ongoing calls when it shuts down and will send an unregistration request to all endpoints. To override this default, set this parameter to `"0"`. This switch is intended mainly for gatekeepers running in direct mode; in routed mode and proxy mode calls will still get disrupted when the gatekeeper shuts down.

- `MaxASNArraySize=400`
  Default: `128`
  Sets the maximum number of elements in an ASN encoded array, eg. the max. number of aliases in a list. Versions of PTLib through v2.10.1 default to 128 elements. Beware of hitting the limits of other vendors if you increase this setting.

- `MaxSocketQueue=10000`
  Default: `100`
  Limit how many bytes to queue for a socket before it is considered dead and will be closed. This probably is only an issue with H.460.17.

- `TTLExpireDropCall=0`
  Default: `1`
  Set whether to drop a registration on TTL expiry even if there appears to be a current call for the endpoint. By default it will force the drop of the call before unregistering the endpoint. A situation may occur if there are bandwidth congestion, RRQ packets are being lost but an active call (likely causing the congestion) is still in progress. By enabling this switch the registration remains current until a RRQ is received or the apparent call either times out or is dropped.

- `MinH323Version=7`
  Default: `2`
  Set the minimum H.225 and H.245 protocol identifiers for gatekeeper generated messages. Usually it is not necessary to set this switch and it is best to leave the version low for interoperability with older endpoints. Use this switch if you are dealing with endpoints that eg. won't enable features when they receive messages with a low version number.

- `RASDiffServ=46`
  Default: `0`
  Set the DiffServ class (DSCP) for RAS messages. (On most Windows versions, setting the the DSCP this way won't work.)

4.6 Section [GkStatus::Auth]

Defines a number of rules regarding who is allowed to connect to the status port. Access to the status port provides full control over your gatekeeper. Ensure that this is set correctly. The status port is active on all IPs GnuGk listens to. You should block as many status ports in your firewall as your setup allows. If you rely on password rules to secure the status port, you should add additional IP based rules ('explicit' or 'regex') to limit the IP range where logins are allowed from.

- `rule=allow`
  Default: `forbid`
  Possible values are
  - `forbid` - disallow any connection.
  - `allow` - allow any connection
  - `explicit` - reads the parameter `ip=value` where `ip` is the IP address of the client, `value` is `1,0` or `allow,forbid` or `yes,no`. If `ip` is not listed the parameter `default` is used.
  - `regex` - the IP of the client is matched against the given regular expression.

    Example:
    To allow client from 195.71.129.0/24 and 195.71.131.0/24:
    `regex=^195\.71\.(129|131)\.[0-9]+$`
  - `password` - the user must provide an appropriate username and password to login. The format of username/password is the same as 8.3 ([SimplePasswordAuth]) section.

  These rules may be combined with "|" (to specify a logical "OR") or "&" (for logical "AND"). For example,
  - `rule=explicit | regex`
    The IP of the client must match `explicit` **or** `regex` rule.
  - `rule=regex & password`
    The IP of the client must match `regex` rule, **and** the user has to login by username and password.

  Using the SSH protocol for the status port implies that all users are authenticated by password, but you can impose additional IP rules eg. "regex & password".

- `default=allow`
  Default: `forbid`
  Only used when `rule=explicit`.

- `DSAKey=/etc/ssh/ssh_host_dsa_key`
  Default: `ssh_host_dsa_key (in current working directory)`
  Path for the file containing the DSA host key. (only used for SSH)

  For SSH access, you must a DSA key or RSA key configured, or both.

  To generate a DSA key (press Enter twice to not set a passphrase)

  ```
  ssh-keygen -t dsa -b 1024 -f ssh_host_dsa_key
  ```

- `RSAKey=/etc/ssh/ssh_host_rsa_key`
  Default: `ssh_host_rsa_key (in current working directory)`
  Path for the file containing the RSA host key. (only used for SSH)

  For SSH access, you must a DSA key or RSA key configured, or both.

  To generate a RSA key (press Enter twice to not set a passphrase)

  ```
  ssh-keygen -t rsa -b 2048 -f ssh_host_rsa_key
  ```

- `Shutdown=forbid`
  Default: `allow`
  To allow the gatekeeper to be shutdown via status port.

- `DelayReject=5`
  Default: `0`
  Time (in seconds) to wait before rejecting an invalid username/password. Useful to insert a delay in brute-force attacks.

4.7 Section [GkStatus::Filtering]

See 13.4 (Status Port Filtering).

4.8 Section [LogFile]

This section defines log file related parameters. Currently, it allows users to specify log file rotation options.

- `Filename=/var/log/gk_trace.log`
  Default: `N/A`
  Set the output filename for the log file (same as -o on the command line). On Windows, backslashes in the file name have to be escaped.

  This setting doesn't change when the config is reloaded!

- `Rotate=Hourly | Daily | Weekly | Monthly`
  Default: `N/A`
  If set, the log file will be rotated based on this setting. Hourly rotation enables rotation once per hour, daily - once per day, weekly - once per week and monthly - once per month. An exact rotation moment is determined by a combination of `RotateDay` and `RotateTime` variables. During rotation, an existing file is renamed to CURRENT_FILENAME.YYYYMMDD-HHMMSS, where YYYYMMDD-HHMMSS is replaced with the current timestamp, and new lines are logged to an empty file. To disable rotation, do not configure the `Rotate` parameter or set it to 0.

  Example 1 - rotate every hour (00:45, 01:45, ..., 23:45):

  ```
  [LogFile]
  Rotate=Hourly
  RotateTime=45
  Filename=/var/log/gk_trace.log
  ```

  Example 2 - rotate every day at 23:00 (11PM):

  ```
  [LogFile]
  Rotate=Daily
  RotateTime=23:00
  Filename=C:\\Logs\\GnuGk.log
  ```

  Example 3 - rotate every Sunday at 00:59:

  ```
  [LogFile]
  Rotate=Weekly
  RotateDay=Sun
  RotateTime=00:59
  ```

Example 4 - rotate on the last day of each month:

```
[LogFile]
Rotate=Monthly
RotateDay=31
RotateTime=23:00
```

5 Routed Mode and Proxy Configuration

5.1 Section [RoutedMode]

Call signaling messages may be passed in two ways: The first method is Direct Endpoint Call Signaling, where call signaling messages are passed directly between the endpoints. The second method is Gatekeeper Routed Call Signaling. With this method, the call signaling messages are routed through the gatekeeper.

When Gatekeeper Routed call signaling is used, there are three different options for routing of the H.245 control channel and media channels.

Case I.

The gatekeeper doesn't route them. The H.245 control channel and media channels are established directly between the endpoints.

Case II.

The H.245 control channel is routed through the gatekeeper, while the media channels are established directly between the endpoints.

Case III.

The gatekeeper routes the H.245 control channel, as well as all media channels, including RTP/RTCP for audio and video, and T.120 channel for data. In this case, no traffic is passed directly between the endpoints. This is usually called a H.323 Proxy, and can be treated as a H.323-H.323 gateway.

This section defines the gatekeeper routed mode options (case I & II). The proxy feature is defined in the 5.2 (next section).

The settings in this section may be updated by reloading the configuration while the gatekeeper is running.

- `GKRouted=1`
  Default: `0`
  Enables gatekeeper routed signaling mode.

- `H245Routed=1`
  Default: `0`
  Enables routing of the H.245 control channel through the gatekeeper. This setting is honored if `GKRouted=1` and H.245 tunneling is disabled for a call. Even when this option is disabled, if Proxy or ProxyForNAT takes effect, a H.245 channel is always routed through the gatekeeper for calls being proxied.

- `CallSignalPort=1721`
  Default: `1720`
  The port for call signaling on the gatekeeper. You may set it to `0` to let the gatekeeper choose an arbitrary port.

- `TLSCallSignalPort=1300`
  Default: `1300`
  The port where GnuGk should listen for TLS (transport layer security) signaling, if enabled in the 12.11 ([TLS] section).

- `CallSignalHandlerNumber=10`
  Default: `5`
  The number of threads dedicated to handle signaling/H.245 channels (between 1-200). You may

increase this number in a heavy loaded gatekeeper. Each thread can process one signaling message at time, so increasing this number will increase call throughput. Under Windows, there exists a default limit of 64 sockets used by a single signaling thread, so each signaling thread is able to handle at most 32 calls (with H.245 tunneling enabled).

- `RtpHandlerNumber=2`
  Default: `1`
  The number of RTP proxy handling threads. Increase this value only if you experience problems with RTP delay or jitter on a heavily loaded gatekeeper. Special care has to be taken on Windows, as RTP handling threads are subject to the same limit of 64 sockets as signaling threads. Each RTP thread is able to handle at most 32 proxied calls (2 sockets per call).

- `AcceptNeighborsCalls=1`
  Default: `1`
  With this feature enabled, the call signaling thread will accept calls without a pre-existing CallRec found in the CallTable, provided an endpoint corresponding to the destinationAddress in Setup can be found in the RegistrationTable, and the calling party is a neighbor or parent gatekeeper. The gatekeeper will also use its own call signaling address in the LCF when responding to the LRQ. Call signaling will be routed to gatekeeper 2 in gatekeeper-to-gatekeeper calls. As a result, the CDRs in gatekeeper 2 will correctly show the connected time, instead of 'unconnected'.

- `AcceptUnregisteredCalls=1`
  Default: `0`
  With this feature enabled, the gatekeeper will accept calls from any unregistered endpoint. Make sure you do proper authentication on these calls if you don't want to let everybody use your gatekeeper. When working with unregistered endpoints, you will probably also want to change the CallSignalPort to 1720.

- `RemoveH245AddressOnTunneling=1`
  Default: `0`
  Some endpoints send h245Address in the UUIE of Q.931 even when h245Tunneling is set to TRUE. This may cause interoperability problems. If the option is TRUE, the gatekeeper will remove h245Address when h245Tunneling flag is TRUE. This enforces the remote party to stay in tunneling mode.

- `RemoveH245AddressFromSetup=1`
  Default: `0`
  With this switch GnuGk will strip H.245 addresses from incoming Setup messages to avoid interoperability issues.

- `DisableH245Tunneling=1`
  Default: `0`
  Force both sides of a call to disable H.245 tunneling.

- `H245TunnelingTranslation=1`
  Default: `0`
  Allow one side of a call to use H.245 tunneling even if the other side does not, with the gatekeeper performing the appropriate H.245 message conversion. This will reduce the number of ports required on the tunneling side of the connection.

- `RemoveCallOnDRQ=0`
  Default: `1`
  With this option disabled, the gatekeeper will not disconnect a call if it receives a DRQ for it. This avoids potential race conditions when a DRQ overtakes a Release Complete. This is only meaningful in routed mode because in direct mode, the only mechanism to signal end-of-call is a DRQ. When using call failover this must be set to 0.

- `DropCallsByReleaseComplete=1`
  Default: `0`
  According to Recommendation H.323, the gatekeeper could tear down a call by sending RAS DisengageRequest to endpoints. However, some bad endpoints just ignore this command. With this option turning on, the gatekeeper will send Q.931 Release Complete instead of RAS DRQ to both endpoints to force them drop the call.

- `SendReleaseCompleteOnDRQ=1`
  Default: `0`
  On hangup, the endpoint sends both Release Complete within H.225/Q.931 and DRQ within RAS. It may happen that DRQ is processed first, causing the gatekeeper to close the call signaling channel, thus preventing the Release Complete from being forwarding to the other endpoint. Though the gatekeeper closes the TCP channel to the destination, some endpoints (e.g. Cisco CallManager) don't drop the call even if the call signaling channel is closed. This results in phones that keep ringing if the caller hangs up before the called number answers. Setting this parameter to `1` makes the gatekeeper always send Release Complete to both endpoints before closing the call when it receives a DRQ from one of the parties.

- `SupportNATedEndpoints=1`
  Default: `0`
  Whether to allow an endpoint behind a NAT box register to the gatekeeper. If yes, the gatekeeper will translate the IP address in Q.931 and H.245 channel into the IP of NAT box.

  GnuGk supports NAT outbound calls (from an endpoint behind NAT to public networks) directly without any necessary modification of endpoints or NAT box. Just register the endpoint with GnuGk and you can make call now.

- `SupportCallingNATedEndpoints=0`
  Default: `1`
  Whether to allow an endpoint behind an NAT box that support GnuGk NAT Traversal technique to receive calls. Use this to block errant gateways that do not support GnuGk Nat Traversal technique properly from causing one way audio problems when trying to call to the gateway. Calls to the gateways return caller unreachable. To be effective SupportNATedEndpoints must be set to 1.

- `TreatUnregisteredNAT=1`
  Default: `0`
  Used in conjunction with AcceptUnregisteredCalls and SupportNATedEndpoints will automatically treat all unregistered calls which cannot be determined as being NAT are treated as being NAT.

  Not all Endpoints send sourceSignalAddress in the setup message which can be used to determine whether a caller is NAT. This adds support to those that don't.

- `ScreenDisplayIE=MyID`
  Default: `N/A`
  Modify the DisplayIE of Q.931 to the specified value.

- `ScreenCallingPartyNumberIE=0965123456`
  Default: `N/A`
  Modify the CallingPartyNumberIE of Q.931 to the specified value.

- `ScreenSourceAddress=MyID`
  Default: `N/A`
  Modify the sourceAddress field of UUIE element from Q.931 Setup message.

- `ForwardOnFacility=1`
  Default: `0`

If yes, on receiving Q.931 Facility with reason **callForwarded**, **routeCallToGatekeeper** or **routeCallToMC**, the gatekeeper will forwards call Setup directly to the forwarded endpoint, instead of passing the message back to the caller. If you have broken endpoints that can't handle Q.931 Facility with reason **callForwarded** (or the other reasons), turn on this option. Note that this feature may not always work correctly, as it does not provide any means of capability renegotiation and media channel reopening. The call is only forwarded if the forwarder is the called party and the H.245 channel is not established, yet.

- `ShowForwarderNumber=0`
  Default: `0`
  Whether to rewrite the calling party number to the number of forwarder. It's usually used for billing purpose. Only valid if `ForwardOnFacility=1`.

- `Q931PortRange=20000-20999`
  Default: `N/A (let the OS allocate ports)`
  Specify the range of TCP port number for Q.931 signaling channels. Note the range size may limit the number of concurrent calls. Make sure this range is wide enough to take into account TIME_WAIT TCP socket timeout before a socket can be reused after closed. TIME_WAIT may vary from 15 seconds to a few minutes, depending on an OS. So if for example your range is 2000-2001 and you made two calls, the next two calls can be made after TIME_WAIT timeout elapses and the sockets can be reused. The same applies to `H245PortRange` and `T120PortRange`. TIME_WAIT can be usually tuned down on most OSes.

- `H245PortRange=30000-30999`
  Default: `N/A (let the OS allocate ports)`
  Specify the range of TCP port number for H.245 control channels. Note the range size may limit the number of concurrent calls. See remarks about TIME_WAIT socket state timeout in the `Q931PortRange` description.

- `SetupTimeout=4000`
  Default: `8000`
  A timeout value (in milliseconds) to wait for a first message (Setup) to be received after a signaling TCP channel has been opened.

- `SignalTimeout=10000`
  Default: `30000`
  A timeout value (in milliseconds) to wait for a signaling channel to be opened after an ACF message is sent or to wait for an Alerting message after a signaling channel has been opened. This option can be thought as a maximum allowed PDD (Post Dial Delay) value.

- `AlertingTimeout=60000`
  Default: `180000`
  A timeout value (in milliseconds) to wait for a Connect message after a call entered Alerting state. This option can be thought as a maximum "ringing time".

- `TcpKeepAlive=1`
  Default: `0`
  Enable/disable keepalive feature on TCP signaling sockets. This can help to detect inactive signaling channels and prevent dead calls from hanging in the call table. For this option to work, you also need to tweak system settings to adjust keep alive timeout. See docs/keepalive.txt for more details. If this switch is not present in the configuration, the socket is left untouched.

- `TranslateFacility=1`
  Default: `0`

Enable this option if you have interoperability problems between H.323v4 and non-H.323v4 endpoints. It converts Facility messages with reason = transportedInformation into Facility messages with an empty body, because some endpoints do not process tunneled H.245 messages inside Facility, if the body is not empty. The conversion is performed only when necessary - if both endpoints are v4 or both endpoints are pre-v4, nothing is changed.

- `FilterEmptyFacility=1`
  Default: `0`
  Filter out Facility messages with reason transportedInformation, but without h245Control or h4501SupplementaryService field. Needed for Avaya interop.

- `SocketCleanupTimeout=1000`
  Default: `5000`
  Define time to wait before an unused socket is closed (if it is not yet closed) and deleted (its memory is released). If you use very small port ranges, like a few ports (e.g. RTPPortRange=2000-2009), you may want to decrease this value to get sockets reusable faster.

- `ActivateFailover=1`
  Default: `0`
  Activate call failover: When activated, GnuGk will try to find other possible routes to a destination if the call fails on the first route. The list of routes for a call is built when the call first comes in and currently not all routing policies are able to provide multiple routes. You can use the 'internal' and the 'sql' policy to provide multiple routes. In addition to that multiple routes can be set by SQL and Radius authenticators.

  For accounting of calls using failover, see the SingleFailoverCDR switch in the 12.1 ([CallTable]) section.

- `FailoverCauses=1-15,21-127`
  Default: `1-15,21-127`
  Define which cause codes in a ReleaseComplete will trigger call failover.

- `DisableRetryChecks=1`
  Default: `0`
  This will disable all checks if a failed call has already received FastStart or H.245 messages. Caution: Using this switch enables you to retry more calls, but you run the risk that some of the retried calls will fail because the caller is already in a state where he can't talk to a new partner.

- `CalledTypeOfNumber=1`
  Default: `N/A`
  Sets Called-Party-Number type of number to the specified value for all calls (0 - UnknownType, 1 - InternationalType, 2 - NationalType, 3 - NetworkSpecificType, 4 - SubscriberType, 6 - AbbreviatedType, 7 - ReservedType).

- `CallingTypeOfNumber=1`
  Default: `N/A`
  Sets Calling-Party-Number type of number to the specified value for all calls (0 - UnknownType, 1 - InternationalType, 2 - NationalType, 3 - NetworkSpecificType, 4 - SubscriberType, 6 - AbbreviatedType, 7 - ReservedType).

- `CalledPlanOfNumber=1`
  Default: `N/A`
  Sets Called-Numbering-Plan of number to the specified value (0 - UnknownType, 1 - ISDN, 3 - X.121 numbering, 4 - Telex, 8 - National standard, 9 - private numbering).

- `CallingPlanOfNumber=1`
  Default: `N/A`
  Sets Calling-Numbering-Plan of number to the specified value (0 - UnknownType, 1 - ISDN, 3 - X.121 numbering, 4 - Telex, 8 - National standard, 9 - private numbering).

- `ENUMservers=e164.org,e164.arpa`
  Default: `N/A`
  Sets the enum server list in priority order separated by (,) for the enum policy. This overrides the PWLIB_ENUM_PATH environmental variable.

- `RDSservers=myvirtualhost.com`
  Default: `N/A`
  Use this to specify a RDS server to query for rds routing policy. This set the domains to use to resolve URI's which do not have SRV records and may be virtually hosted or where SRV records are stored in another host. This overrides the PWLIB_RDS_PATH environmental variable.

- `CpsLimit=10`
  Default: `0`
  Limit the rate of incoming calls to n calls per second. If more calls are received they are rejected at the TCP level without H.323 error messages, so they won't show up in CDRs. A value of zero (default) disables the feature.

  The limit only applies if the calls in the check interval are greater than check-interval * CPS-rate. This allows small call spikes on a machine without much load, but will apply strict limits when the overall load is high.

  This feature is meant to shield the gatekeeper from overload and to avoid as much resource usage a possible in an overload situation.

  Currently the calls are blocked when the first message comes in on the signaling port. This makes it very effective for unregistered calls. However, ARQs are not blocked, so it will be less effective with registered calls.

- `CpsCheckInterval=1`
  Default: `5`
  Define the check interval in seconds before the CpsLimit is applied.

- `GenerateCallProceeding=1`
  Default: `0`
  When set, GnuGk will generate a CallProceeding for each Setup message it receives. This can be helpful to avoid a timeout in calling endpoints if the destination takes a long time to answer or the call is processed in a virtual queue. Without setting UseProvisionalRespToH245Tunneling=1 this will disable H.245 tunneling.

  CallProceeding messages sent by endpoints or gateways will be translated into Facility or Progress messages.

- `UseProvisionalRespToH245Tunneling=1`
  Default: `0`
  WARNING: This is an experimental feature and not tested very well.

  If you only use H.323 equipment that supports provisionalRespToH245Tunneling, you can set this switch to keep H.245 tunneling enabled when using gatekeeper generated CallProceeding.

- `EnableH450.2=1`
  Default: `0`
  When set, GnuGk will intercept H.450.2 call transfer messages and if possible transfer the call on behalf

of the endpoint. This allows the endpoint initiated transferring of calls where the remote endpoint may not support H.450 and the gatekeeper initiates the call transfer.

- `H4502EmulatorTransferMethod=Reroute`
  Default: `callForwarded`
  Set the call transfer method for the H.450.2 emulator. It defaults to sending a callFordwarded Facility to the endpoint. Setting it to "Reroute" uses a gatekeeper based TCS=0 transfer. ("Reroute" is still considered and experimental feature, that should be used with care.)

- `TranslateReceivedQ931Cause=17:=34`
  Default: `N/A`
  Translate all received cause codes in ReleaseComplete messages. In the above example code 17 (User busy) will be translated into cause code 34 (No circuit/channel available).

- `TranslateSentQ931Cause=21:=34,27:=34`
  Default: `N/A`
  Translate all cause codes in ReleaseComplete messages sent out. In the above example code 21 and 27 will be translated into cause code 34, because this particular gateway might deal with error code 34 better than with others.

- `RemoveH235Call=1`
  Default: `0`
  For compatibility with endpoints which do not support large Setup messages or if endpoints send incorrect H.235 tokens, this switch removes all clearTokens and cryptoTokens from Setup and Connect messages.

  If you turn the feature on with setting the switch to 1, the H.235 tokens will be removed from all calls. You can also specify a list of networks, the only calls from these networks get the H.235 tokens removed, eg. RemoveH235Call=192.168.1.0/24, 10.0.1.0/32.

- `RemoveH460Call=1`
  Default: `0`
  For compatibility with pre-H323v4 devices that do not support H.460, this switch strips the H.460 feature advertisements from the Setup PDU. Usually they should be ignored anyway; use this switch if they cause trouble.

- `ForceNATKeepAlive=1`
  Default: `0`
  Force ALL registrations to use a keepAlive TCP socket.

- `EnableH46017=1`
  Default: `0`
  Enable support for H.460.17. To enable H.460.19 for the media stream, you should also set EnableH46018=1.

- `EnableH46018=1`
  Default: `0`
  Enable support for H.460.18 and H.460.19. This feature is covered by patents held by Tandberg. If you don't use the official releases by the GNU Gatekeeper Project, make sure you have a valid license before enabling it.

- `H46018NoNat=0`
  Default: `1`
  Enable H.460.18 even if the endpoint is not behind a NAT. Setting to 0 will disable H.460.18 if the endpoint is detected as not being behind a NAT. If H.460.23 is supported and enabled then direct media is still supported.

- `EnableH46023=1`
  Default: `0`
  Enable support for H.460.23/.24. You must also set STUN servers for H.460.23/.24 to become active.

- `H46023STUN=stun.ekiga.net,192.168.1.10`
  Default: `N/A`
  Sets the STUN server list for use with H.460.23 separated by (,). Each Network interface must have a STUNserver set for H.460.23 support on that interface.

- `H46023PublicIP=1`
  Default: `0`
  Newer endpoints on public IP addresses can receive calls from endpoints behind NAT. This feature when enabled will presume all endpoints that are not NAT can receive calls from endpoints behind NAT for the purpose of H.460.24 media pathway calculations so to avoid proxying of media. This maybe used in conjunction with AlwaysRewriteSourceCallSignalAddress=0 to trick the remote endpoint to think that the call is coming direct from behind NAT and not routed via the gatekeeper.

- `H46023SignalGKRouted=1`
  Default: `0`
  Force all call signaling for NAT to be GK routed. There are a number of conditions where call signaling may be offloaded when using H.460.23/.24 This switch will force all the signaling to be Gatekeeper routed.

- `H46024ForceDirect=1`
  Default: `0`
  Force all media to NOT proxy if the remote NAT status cannot be determined. Most (not all) H.323 devices are able if on a public IP to receive calls from endpoints that are behind NAT. Use this switch with caution.

- `H46024ForceNat=1`
  Default: `0`
  Where an endpoint is detected as being on the public internet force the device to appear as being firewalled. This resolve inconsistent behaviour where firewalled endpoints on public IP appear not to be firewalled.

- `NATStdMin=18`
  Default: `N/A`
  Require registering endpoints detected as being behind a NAT to support a Standard NAT Traversal mechanism. When an endpoint registers from behind a NAT and does not support the minimum NAT standard then the registration will be rejected with a reason neededFeatureNotSupported. Valid values are "18" for H.460.18/.19 and "23" for H.460.23/.24

- `EnableH46026=1`
  Default: `0`
  Enable support for H.460.26 (media over TCP).

- `UseH46026PriorityQueue=0`
  Default: `1`
  Use a priority queue when sending to H.460.26 endpoints. It will batch RTP packets together and make sure the endpoint isn't flooded with more messages than it can handle.

- `TranslateSorensonSourceInfo=1`
  Default: `0`
  Translate the non-standard caller information eg. from a Sorenson VP200 into sourceAddress and CallingPartyIE.

- `RemoveSorensonSourceInfo=1`
  Default: `0`
  Remove the non-standard caller information eg. from a Sorenson VP200 after translation.

- `RemoveFaxUDPOptionsFromRM=1`
  Default: `0`
  An Avaya Communication Manager 3.1 system equipped with TN2602AP media processors becomes confused when it receives t38FaxUdpOptions in t38FaxProfile of H.245 RequestMode. AddPac VoiceFinder is an example of an application which does this. At that point, the TN2602AP will begin to send larger T.38 packets than the receiver can process, resulting in facsimile document distortion. This switch will remove t38FaxUdpOptions from RequestMode, making the combination of Avaya Communication Manager 3.1 equipped and TN2602AP media processors compatible with endpoints which send t38FaxUdpOptions in RM.

- `AlwaysRewriteSourceCallSignalAddress=0`
  Default: `1`
  When set to false or 0, GnuGk will not rewrite the sourceCallSignalAddress to its own IP in routed mode. This helps some endpoints to get through NATs. In proxy mode, the IP is always rewritten to GnuGk's IP, regardless of this switch.

- `AutoProxyIPv4ToIPv6Calls=0`
  Default: `1`
  Automatically put calls between different IP versions into full proxy mode. Note that this auto detection only looks at the call signal addresses to make the decision. It is possible that one endpoint decides to use H.245 or media IPs with a different IP version later on and the call will fail if the receiving endpoint isn't capable of handling multiple IP versions.

- `EnableH235HalfCallMedia=1`
  Default: `0`
  When the endpoint on one side of a call supports encryption and the endpoint on the other side does not, the gatekeeper will act as a "man-in-the-middle" and encrypt the media stream to the encryption-capable system. A decrypted media stream will be sent to the endpoint which is otherwise unable to encrypt / decrypt traffic because of licensing issues, lack of encryption chip support in the hardware, obsolescence, etc. This may be useful if the system you are trying to reach is on the Internet; your internal traffic can remain unencrypted, but your external traffic will be secure.

  Enabling this feature will force call signaling for all calls to routed-mode, and will set it to proxy-mode for encrypted calls.

  When not using RTP multiplexing, the caller and called endpoint must be on different IPs and may not be behind the same NAT. The endpoints also must send RTP from the same IP as their signalling messages.

  As of Version 3.x of GnuGk, encryption of data channels is not supported.

- `RequireH235HalfCallMedia=1`
  Default: `0`
  Require at least one leg of the call to be encrypted. (Terminate the call if both legs are unencrypted.)

- `H235HalfCallMaxTokenLength=2048`
  Default: `1024`
  Set the maximum token length for for H.235 half call media. With 1024 bit tokens AES 128 encryption will be used. For token length greather than 1024 GnuGk will use AES 256.

- `EnableH235HalfCallMediaKeyUpdates=1`
  Default: `0`

EXPERIMENTAL: Update media keys after they have been used for too many operations to remain cryptographically safe. This feature has only been tested GnuGk to GnuGk, it seems endpoints from most vendors do not support key updates as defined in H.235.6.

- `Q931DecodingError=Drop`
  Default: `Disconnect`
  Specify GnuGk's reaction to invalid Q.931 messages that it cannot decode. Until version 3.1 GnuGk would "Disconnect" the connection to protect internal endpoints from possibly malicious messages, but if you have some buggy endpoints that you can't get fixed, it might be helpful just to "Drop" this Q.931 message that couldn't be decoded. The last option to simply "Forward" the messages should be used with great care, since invalid messages might cause your endpoints to crash or worse.
- `PregrantARQ=1`
  Default: `0`
  Use pre-ganted ARQ model: Endpoints don't have to send ARQ before a call and will save one message round-trip in the call establishment. Endpoints that don't support this H.323 version 2 feature and will keep sending ARQs as usual.

  Note: When using this switch in a direct-mode configuration, you will loose almost all control over your calls. When the gatekeeper is in routed-mode, calls without ARQ can still be authenticated on the Setup message.
- `EnableH460P=1`
  Default: `0`
  WARNING: This is an experimental feature to support the not-yet-released H.460 presence standard.
- `ProxyHandlerHighPrio=0`
  Default: `1`
  Set the proxy handler for signalling connections to high priority. In some virtual server configurations we have to turn this off if PTLib fails with "pthread_setschedparam failed". This switch will go away once we can detect this condition automatically.
- `H225DiffServ=46`
  Default: `0`
  Set the DiffServ class (DSCP) for H.225 messages. (On most Windows versions, setting the the DSCP this way won't work.)
- `H245DiffServ=46`
  Default: `0`
  Set the DiffServ class (DSCP) for H.245 messages. (On most Windows versions, setting the the DSCP this way won't work.)

5.2 Section [Proxy]

The section defines the H.323 proxy features. It means the gatekeeper will route all the traffic between the calling and called endpoints, so there is no traffic between the two endpoints directly. Thus it is very useful if you have some endpoints using private IP behind an NAT box and some endpoints using public IP outside the box.

The gatekeeper can do proxy for logical channels of RTP/RTCP (audio and video) and T.120 (data). Logical channels opened by fast-connect procedures or H.245 tunneling are also supported.

Note to make proxy work, the gatekeeper must have **direct connection** to both networks of the caller and callee.

- `Enable=1`
  Default: `0`
  Whether to enable the proxy function. You have to enable gatekeeper routed mode first (see the 5.1 (previous section)). You don't have to specify H.245 routed. It will automatically be used if required.

- `InternalNetwork=10.0.1.0/24`
  Default: `N/A`
  If you want to override automatic detection of networks behind the proxy, you may do so by specifying them here. Multiple internal networks are allowed. Packets to internal networks will use the local interface as sender instead of the default IP or ExternalIP. For internal networks, the proxying can be disabled, even when global proxying is activated.

  Format:

  `InternalNetwork=network address/netmask[,network address/netmask,...]`

  The netmask can be expressed in decimal dot notation or CIDR notation (prefix length), as shown in the example.

  Example:

  `InternalNetwork=10.0.0.0/255.0.0.0,192.168.0.0/24`

- `ProxyAlways=1`
  Default: `0`
  Always proxy all calls regardless of other settings.

- `T120PortRange=40000-40999`
  Default: `N/A (let the OS allocate ports)`
  Specify the range of TCP port number for T.120 data channels. Note the range size may limit the number of concurrent calls. See remarks about TIME_WAIT socket state timeout in the `Q931PortRange` description.

- `RTPPortRange=50000-59999`
  Default: `1024-65535`
  Specify the range of UDP port number for RTP/RTCP channels. Since RTP streams require two sockets, the range must contain an even number of ports. Note that the size of the specified range may limit the number of possible concurrent calls.

- `ProxyForNAT=1`
  Default: `1`
  If set, the gatekeeper will function as a proxy for calls where one of the participating endpoints is behind a NAT box. This ensures the RTP/RTCP stream can penetrate into the NAT box without modifying it. However, the endpoint behind the NAT box must use the same port to send and receive RTP/RTCP stream. If you have bad or broken endpoints that don't satisfy the precondition, you should disable this feature and let the NAT box forward RTP/RTCP stream for you.

- `ProxyForSameNAT=1`
  Default: `1`
  Whether to proxy for calls between endpoints from the same NAT box. There is a degree of uncertainty when endpoints are behind the same NAT as to whether they can communicate directly as one or both may be on subNATs. Disable this feature with caution.

- `DisableRTPQueueing=0`
  Default: `1`
  Sometimes GnuGk will receive RTP data before it knows where to forward it to. GnuGk can buffer this data up to a certain amount and send it once the destination becomes available. In some cases this can cause a short loopback of RTP data.

- `EnableRTPMute=1`
  Default: `0`
  This setting allows either call party in media proxy mode to mute the audio/video by sending a * as either string or tone.userinput. The sending of * mutes the audio/video and a subsequent send of * unmutes the audio/video. Only the party who muted the call can unmute. This is designed as a hold function for terminals which do not support H450.4.

- `EnableRTCPStats=1`
  Default: `0`
  When enabled, GnuGk will collect RTCP sender reports and send them to the Radius server.

- `RemoveMCInFastStartTransmitOffer=1`
  Default: `0`
  Remove the mediaChannel from fastStart transmit offers. For unicast transmit channels, mediaChannel should not be sent on offer; it is the responsibility of the callee to provide mediaChannel in an answer.

- `SearchBothSidesOnCLC=1`
  Default: `0`
  The H.245 CloseLogicalChannel request should only reference the endpoint's own logical channels. Some bad endpoint implementations require searching and closing logical channels for the other endpoint as well. Up to version 2.3.0 GnuGk did this automatically, but it can break channel establishment in some cases, so you must enable this switch if you have these broken endpoints.

- `CheckH46019KeepAlivePT=0`
  Default: `1`
  Verify the correct payload type on H.460.19 keep alive packets. Disable for endpoints advertising an incorrect payload type.

- `RTPMultiplexing=1`
  Default: `0`
  Enable H.460.19 RTP multiplexing. H.460.19 must be enabled for multiplexing.

  NOTE: To change RTP multiplexing settings, including ports, you must restart GnuGk. A configuration reload will not re-read this configuration item.

- `RTPMultiplexPort=4000`
  Default: `3000`
  Set the RTP port for H.460.19 RTP multiplexing.

- `RTCPMultiplexPort=4001`
  Default: `3001`
  Set the RTCP port for H.460.19 RTP multiplexing.

- `RTPDiffServ=46`
  Default: `4`
  Set the DiffServ class (DSCP) for proxied RTP. The default value corresponds to the old IPTOS_LOWDELAY flag we have used previously. New installations should use eg. 46 which is DSCP EF reccomended for RTP. For IPv6 packets the TCLASS is set. (On most Windows versions, setting the the DSCP this way won't work.)

- `ExplicitRoutes=10.2.1.5/16,10.6.1.3/16,11.0.0.0/8-20.1.1.1`
  Default: `n/a`
  Add explicit routing rules to GnuGk's internal routing table. Rules can have 2 formats: `sourceIP/mask` or `network/mask-sourceIP`. The above example would use 10.2.1.5 as sender IP for all messages to the 10.2.0.0/16 network and 10.6.1.3 for messages to the 10.6.0.0/16 network. Messages to the 11.0.0.0/8 network get 20.1.1.1 as sender IP. All sender IPs should be included in the list of Home IPs.

- `IgnoredSignaledIPs=1`
  Default: `0`
  EXPERIMENTAL: Ignore all IPs for RTP streams signaled by the endpoints and rely 100% on port auto detection. In some cases this results in much batter NAT traversal for unregistered endpoints and endpoints not capable of a NAT traversal protocol. This feature gets automatically disabled for calls from endpoints using H.460 NAT traversal and for H.239 video streams which are unidirectional by nature so we can't use auto detection.

- `IgnoredSignaledPrivateH239IPs=1`
  Default: `0`
  EXPERIMENTAL: Also ignore IPs signaled for H.239 streams if they are private IPs.

5.3 Section [ModeSelection]

In routed mode or proxy mode, you may use this section to specify the exact routing mode (routed mode, routed mode plus H.245 routing or proxy mode) on a per-IP network basis.

Syntax:

```
network=mode[,mode]
```

The network is specified by an IP plus optional CIDR, eg. 192.168.1.0/24. The rule for the network with the longest netmask is used (the most specific).

Possible modes are (the names are case in-sensitive)

- `ROUTED`
  Routed mode where Q.931 messages are routed, but not H.245 messages (unless H.245 tunneling is active).

- `H245ROUTED`
  Routed mode plus H.245 routing.

- `PROXY`
  Proxy mode with RTP proxying.

The first mode is used for calls into and out of the specified network. The second mode is used for calls that stay inside the network. If only one mode is specified it is used for both cases.

Example:

In this example calls into and out of the 1.2.3.0/24 network are proxied, but calls that remain inside this network are in routed mode. Calls in the 3.4.5.0/24 are always proxied, even when they remain inside the network, unless IP 3.4.5.6 is involved. If 2 networks have a rule for the call, the one with the most proxying is used, eg. a call from 192.168.1.222 to 3.4.5.20 would be proxied.

```
[ModeSelection]
127.0.0.0/24=ROUTED
192.168.0.0/18=H245ROUTED,ROUTED
1.2.3.0/24=PROXY,ROUTED
3.4.5.0/24=PROXY,PROXY
3.4.5.6=ROUTED
2005:4dd0:ff00:99a::9/64=PROXY
```

If no rules match the settings then [RoutedMode]GkRouted=, H245Routed= or [Proxy]Enable= are used to determine the routing mode.

There are a few cases where these rules don't apply, because *the GNU Gatekeeper* knows that the call needs proxying: For example calls involving H.460.18/.19 will always be proxied (because this protocol requires proxying).

p>

5.4 Section [ModeVendorSelection]

In routed mode or proxy mode, you may use this section to specify the exact routing mode (routed mode, routed mode plus H.245 routing or proxy mode) on vendor specific basis. The vendor information is collected from the H225_EndpointType field of the setup and connect message

Syntax:

```
vendor=mode
```

The vendor is specified by an string matching value. The rule for the longest string match is used (the most specific).

Possible modes are in accordance with the [ModeSelection] section above.

Example:

```
[ModeSelection]
VoIP Technologies=PROXY
```

6 Routing Configuration

The following sections in the config file can be used to configure how calls are routed.

For GnuGk, "routing" means that the gatekeeper must find a destination IP for each new call.

For example GnuGk may need to decide where to send a voice call given a particular E.164 destination; there may be multiple IP-to-ISDN gateways which it may choose from for that E.164 address.

Routing decisions are typically made by examining the called name or number, but GnuGk has flexibility in what it evaluates in order to successfully route the call.

Each call gets passed down a chain of routing policies. Each policy may route the call and terminate the chain or modify it and pass it on. You can use the setting in the following sections to specify which policies to use and modify their behavior.

6.1 Section [RoutingPolicy]

This section explains how *GNU Gatekeeper* routing policies are configured.

An incoming call request can be routed using the following methods:

- `explicit`
  The destination is explicitly specified in the call to be routed. This policy is needed for dialing by IP address. You can define mappings for the destination IP in the 6.10 (Routing::Explicit) section.
- `internal`
  The classic rule; search for the destination in RegistrationTable
- `parent`
  Route the call using information sent by the parent gatekeeper in reply to an ARQ the gatekeeper will send (only LRQs to the child will be forwarded as LRQs). You can define your parent gatekeeper using the 12.5 (Endpoint) section.
- `neighbor`
  Route the call using neighbors by exchanging LRQ messages.
- `dns`
  The destination is resolved using DNS A records or IP addresses in the called alias. This policy can be configured in the 6.6 (Routing::DNS) section.
- `sql`
  Route calls by rewriting the called alias with a database query or send them directly to a destination IP. The database parameters are specified in the 6.11 (Routing::Sql) section.
- `ldap`
  Route calls by looking up the called alias in an LDAP server (searching the H323ID and TelephoneNo attribute) and send the call to the IP in the CallDestination attribute.

  The LDAP server is configured in the 8.14 (GkLDAP::Settings) section and the attributes are defined in the 8.15 (GkLDAP::LDAPAttributeNames) section.
- `vqueue`
  Use the virtual queue mechanism and generate a RouteRequest event to allow an external application to make a routing decision.

- `numberanalysis`
  Provides support for overlapped digit sending for ARQ messages. This also partially supports Setup messages (no overlapped sending - only number length validation).

- `enum`
  ENUM (RFC3761) is a method to use DNS lookups to convert real International Direct Dialing E.164 numbers into H.323 dialing information. The default servers are `e164.voxgratia.net`, `e164.org` and `e164.arpa`. To specify your own list of servers use the `ENUMservers` switch in the RoutedMode section.

  The enum policy replaces the destination with the information returned by the ENUM server, so you must have the appropriate routing policies to continue processing the call after the enum policy. You should have the srv and dns policies after the enum policy, because the new location is often returned in the form of 'number@gatekeeper' and the srv and dns policies are needed to resolve this.

  Finally, keep in mind that each routing check with the enum policy requires a DNS lookup. To speed up your routing, make sure you resolve internal destinations before the enum policy is applied.

  This policy can be configured in the 6.7 (Routing::ENUM) section.

  Additional ENUM schemas for gateways aside from the default "E2U+h323" may be supported via the "enum::id" Routing policy refer 6.7 (Routing::ENUM) section.

- `srv`
  DNS SRV or H.323 Annex O allows for the routing of calls using a H.323 URI. Addresses can be configured as user (at) domain. H.323 URIs are stored in the SRV DNS records of the domain and are queried to find the destination.

  This policy can be configured in the 6.8 (Routing::SRV) section.

  Additional SRV schemas for gateways aside from the default "h323ls._udp." and "h323cs._tcp." may be supported via the "srv::id" Routing policy refer 6.8 (Routing::SRV) section.

- `rds`
  RDS Resolver Discovery Service or DDDS Dynamic Delegation Discovery Service (examples in RFC3404 sect 5.2/5.3) This policy is a mechanism whereby domain name SRV records are hosted in central DNS servers. The servers are set by [RoutedMode] RDSServers and are queried in order to resolve H323+D2U NAPTR records which contain H.323 Annex O SRV records for domains. This can be used to virtually host URL domains or centralize the control of SRV records.

  This policy can be configured in the 6.9 (Routing::RDS) section.

- `forwarding`
  This policy will perform a database lookup if calls to this destination should be forwarded. The configuration for this policy must be in the 6.14 (Routing::Forwarding) section.

- `catchall`
  This policy will route all calls that reach it to one endpoint specified in the 6.15 (Routing::CatchAll) section. You can use it as a fallback at the end of the policy chain to route all calls which would otherwise fail.

- `lua`
  This policy runs the LUA script defined in 6.15 (Routing::Lua) section to set a call destination.

- `neighborsql`
  Same as neighbor policy except the target for the LRQ messages are retrieved from a database. The database parameters are identical to the sql routing policy.

- `uriservice`
  Apply a routing policy based on the URI schema eg xmpp:me@mydomain.com or xmpp:192.168.1.1.

The schemas is defined in 6.17 (Routing::URIService) section. If schema is an IP address will return the [Routing::URIService] gateway setting. This can be used chained with [Routing::ENUM::schema] and [Routing::SRV::schema] to completely resolve addresses.

Default configuration for routing policies is as follows:

```
   [RoutingPolicy]
default=explicit,internal,parent,neighbor
```

If one policy does not match, the next policy is tried.

These policies can be applied to a number of routing request types and routing input data. The different types are ARQ, LRQ, Setup and Facility (with the callForwarded reason). There is also the general routing policy, which is a default for the other types.

Example:

```
   [RoutingPolicy]
h323_ID=dns,internal
002=neighbor,internal
Default=internal,neighbor,parent
```

When a message is received which requires a routing decision, all calls to an alias of the h323_ID type will be resolved using DNS. If DNS fails to resolve the alias, it is matched against the internal registration table. If a call is requested to an alias starting with 002, the neighbors will be checked first, then the internal registration table. If the requested alias is not an h323_ID or an alias starting with 002, the default policy is used by querying the internal registration table, then the neighbors, and if those fail, the parent.

Routing policies are applied to the first message of a call: The ARQ for calls from registered endpoints, the Setup for calls from unregistered endpoints, the LRQ for calls from neighbors and certain Facility messages for calls that are forwarded by GnuGk using the ForwardOnFacility feature. You can specify different routing policies for each type of call by using the [RoutingPolicy::OnARQ], [RoutingPolicy::OnLRQ], [RoutingPolicy::OnSetup] and [RoutingPolicy::OnFacility] sections using the same syntax explained above.

Example:

```
   [RoutingPolicy::OnARQ]
default=numberanalysis,internal,neighbor
```

A typical ENUM routing setup would look like this:

Example:

```
   [RoutingPolicy]
default=explicit,internal,enum,srv,dns,internal,parent,neighbor
```

6.2 Section [RasSrv::RewriteE164]

This section defines the rewriting rules for dialedDigits (E.164 number).

Format:

`[!]original-prefix=target-prefix`

If the number begins with `original-prefix`, it is rewritten to `target-prefix`. If the '`!`' flag precedes the `original-prefix`, the sense is inverted and the target-prefix is prepended to the dialed number. Special wildcard characters (`'.'` and `'%'`) are available.

Example:

`08=18888`

If you dial `08345718`, it is rewritten to `18888345718`.

Example:

`!08=18888`

If you dial `09345718`, it is rewritten to `1888809345718`.

Option:

- `Fastmatch=08`
  Default: `N/A`
  Only rewrite dialDigits beginning with the specified prefix.

6.3 Section [RasSrv::RewriteAlias]

This section defines the rewriting rules for aliases. This can be used to map gatekeeper assigned aliases to registered endpoints.

Format:

`[!]original-alias=target-alias`

If the alias is `original-alias`, it is rewritten to `target-alias`.

Example:

`bill=033123456`

6.4 Section [RasSrv::GWRewriteE164]

This section describes rewriting the dialedDigits E.164 number depending on the gateway a call has come from or is being sent to. This allows for more flexible manipulation of the dialedDigits for routing etc.

Despite the name of the section, you can not only rewrite calls from and to gateways, but also calls from terminals (regular endpoints) and neighbor gatekeepers.

In combination with the 6.3 (RasSrv::RewriteE164) you can have triple stage rewriting:

```
Call from "gw1", dialedDigits 0867822
                |
                |
                V
Input rules for "gw1", dialedDigits now 550867822
                |
                |
                V
```

```
Global rules, dialedDigits now 440867822
                |
                |
                V
Gateway selection, dialedDigits now 440867822, outbound gateway "gw2"
                |
                |
                V
Output rules for "gw2", dialedDigits now 0867822
                |
                |
                V
Call to "gw2", dialedDigits 0867822
```

Format:

`alias=in|out=[!]original-prefix=target-prefix[;in|out...]`

If the call matches the alias, the direction and begins with `original-prefix` it is rewritten to `target-prefix`. If the '`!`' flag precedes the `original-prefix`, the sense is inverted. Special wildcard characters (`'.'` and `'%'`) are available. '.' matches one character and '%' matches any number of characters. Multiple rules for the same gateway are separated by ';'.

To convert dialed digits into post dial digits that are sent to the remote side after the call connects as UserInputIndications, use 'I' (for Input) on the prefix side and 'P' (for Postdial) on the target side. Please note that H.245 routing through the gatekeeper must be active to send post dial digits.

Calls from and to gateways and terminals are matched by their first alias. Calls from and to neighbors are matched by the neighbor ID in the GnuGk config (the XXX in the [Neighbor::XXX] section name) or the gatekeeper identifier of the neighbor if it is set.

Note that when you have multi-homed neighbors or are accepting non-neighbor LRQs, the source of the call can not always be determined and no IN rule for a neighbor will match. In these cases you should only use OUT and [RasSrv::RewriteE164] rules.

Example:

`gw1=in=123=321`

If a call is received from "gw1" to `12377897`, it is rewritten to `32177897` before further action is taken.

Post Dial Example:

`gw1=out=09III=09PPP`

If a call is sent out through "gw1" to `09123`, it is rewritten to `09` and `123` are sent as post dial digits.

Neighbor Example 1:

In this example the neighbor is identified by its ID and incoming calls from NbGk will have their 01 prefix replaced by a 04 prefix. Outgoing calls will have 04 replaced with 01.

```
[RasSrv::Neighbors]
NbGk=GnuGk

[Neighbor::NbGk]
GatekeeperIdentifier=GK-PW-Prox
Host=192.168.1.100
SendPrefixes=*
AcceptPrefixes=*
```

```
[RasSrv::GWRewriteE164]
NbGk=in=01=04;out=04=01
```

Neighbor Example 2:

In this example the neighbor is identified by its gatekeeper identifier and incoming calls from GK-PW-Prox that don't have a 0049 prefix get the prefix prepended. A call to "1234" would be rewritten to "00491234", while a call to "00496789" would proceed unchanged because the "If incoming does not start with 0049 and any number of digits after 0049, then prepend 0049" logic would be false (because we already have 0049 at the beginning.)

```
[RasSrv::Neighbors]
NbGk=GnuGk

[Neighbor::NbGk]
GatekeeperIdentifier=GK-PW-Prox
Host=192.168.1.100
SendPrefixes=*
AcceptPrefixes=*

[RasSrv::GWRewriteE164]
GK-PW-Prox=in=!0049.=0049.
```

6.5 Section [Endpoint::RewriteE164]

Once you specify prefix(es) for your gatekeeper endpoint, the parent gatekeeper will route calls with **dialed-Digits** beginning with that prefixes. The child gatekeeper can rewrite the destination according to the rules specified in this section. By contrast, when an internal endpoint calls an endpoint registered to the parent gatekeeper, the source will be rewritten reversely.

Format:

```
external prefix=internal prefix
```

For example, if you have the following configuration,

```
                        [Parent GK]
                        ID=MasterGK
                        /         \
                       /           \
                      /             \
                     /               \
                [Child GK]          [EP3]
                ID=ProxyGK          E164=18888200
                Prefix=188886
                /       \
               /         \
              /           \
           [EP1]         [EP2]
           E164=601      E164=602
```

With this rule:

```
188886=6
```

When EP1 calls EP3 by 18888200, the CallingPartyNumber in the Q.931 Setup will be rewritten to 18888601. Conversely, EP3 can reach EP1 and EP2 by calling 18888601 and 18888602, respectively. In consequence, an endpoint registered to the child gatekeeper with prefix '6' will appear as an endpoint with prefix '188886', for endpoints registered to the parent gatekeeper.

The section does not relate to the section 6.3 (RasSrv::RewriteE164), though the latter will take effect first.

6.6 Section [Routing::DNS]

- `ResolveNonLocalLRQ=0`
  Default: `1`
  This switch determines whether the DNS policy should resolve hostnames or IPs in LRQs that don't terminate locally.

6.7 Section [Routing::ENUM]

- `ResolveLRQ=1`
  Default: `0`
  This switch toggles whether the 'enum' policy should resolve LRQs.

Additional ENUM schemas may be configured by the [Routing::ENUM::id]

Format:

```
<enum schema>=<protocol gateway>
```

Example:

```
      [Routing::ENUM::2]
    E2U+xmpp=mygateway@mydomain.com
```

6.8 Section [Routing::SRV]

- `ResolveNonLocalLRQ=1`
  Default: `0`
  This switch selects if the 'srv' policy should resolve hostnames in LRQs that don't terminate locally.

Additional SRV schemas may be configured by the [Routing::SRV::id]

Format:

```
<SRV schema>=<protocol gateway>[;default schema port]
```

Example:

```
      [Routing::SRV::2]
    _xmpp-server._tcp=mygateway@mydomain.com
```

6.9 Section [Routing::RDS]

- `ResolveLRQ=1`
  Default: `0`
  This switch selects if the 'rds' policy should resolve hostnames in LRQs.

6.10 Section [Routing::Explicit]

You can define a mapping where calls to certain IPs should be routed by the 'explicit' policy. The new destination can either be another IP or an alias destination of any type. If you rewrite the destination to something other than an IP, make sure you have other routing policies in the chain behind the 'explicit' policy that can handle the new destination.

Format:

```
IP=newIP[:port] | E.164 | alias
```

Example:

```
   [Routing::Explicit]
192.168.1.100=10.10.1.100
192.168.1.101=10.10.1.101:1720
192.168.1.102=654251
192.168.1.103=peter
192.168.1.104=joe@company.com
```

6.11 Section [Routing::Sql]

Rewrite the called alias with a SQL query. Supports routing OnARQ, OnLRQ and OnSetup.

If the string returned from the database is 'REJECT' (upper or lower case), the call is rejected. If the string matches a dotted IP address, it is taken as destination IP otherwise it is treated as a new destination alias. If 2 columns are returned, the first is treated as the new destination alias and the second is treated as new destination IP. If the 2nd column contains 'IGNORE', the database result is treated as if it would only contain 1 result column. (This allows simpler SQL queries in some cases.)

If multiple rows of destination IPs are returned they are used as alternative routes for failover and GnuGk will try them in order.

When at least one destination IP is specified or the call is rejected, the SQL policy will end the routing chain. If only the alias is changed, the chain continues with this updated alias.

When rejecting a call, the 2nd column can contain an integer designating the reject reason (H.225 AdmissionRejectReason for registered calls, H.225 LocationRejectReason for neighbor calls, H.225 disconnect reason for unregistered calls).

If the database returns nothing, the call is passed on unchanged.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `Query=SELECT ...`
  Default: `N/A`
  Define a SQL query to fetch the new destination number. The query is parameterized - that means parameter replacement is made before each query is executed. The following parameters are defined:
  - `%c` - the called alias
  - `%p` - the called IP (only available on Setup, empty otherwise)
  - `%s` - the calling IP
  - `%r` - the calling aliases
  - `%{Calling-Station-Id}` - the calling station ID (same value as used in accounting and authentication events)

  - `%i` - the call ID
  - `%m` - the message type (ARQ, LRQ or Setup)
  - `%{client-auth-id}` - a 64 bit integer ID provided to GnuGk when authenticating the call (through SQLAuth)
  - `%{language}` - language if available

  Some of these can be empty if they aren't included in the ARQ, LRQ or Setup message.

  If the query returns no rows, the current alias is used. Otherwise, the first result row is used.

  Query string examples. Note that these are examples; the actual structure and schema are user defined, as are the various field names in these examples. GnuGk is simply expecting either IP addresses or aliases as a result of the query.

```
SELECT destination FROM routes WHERE called = '%c'
SELECT concat(prefix,'%c') FROM routes WHERE prefix = LEFT('%c', 5)
SELECT gatewayip FROM routes WHERE prefix = LEFT('%c',5)
SELECT concat(prefix,'%c'), gatewayip FROM routes WHERE route = LEFT('%c', 5) limit 3
```

- `EnableRegexRewrite=1`
  Default: `0`
  Enable basic regex rewriting where parts of the original called alias are inserted into the result of the database query.

  Regular expressions:

  - {\1} - all of the original called alias
  - {^\d(4)} - first 4 digits
  - {\d(4)$} - last 4 digits

  Examples:

  Assuming the called alias was "12345678" and the database returns "{\1}@my.com" then all character are inserted so the new destination is "1234578@my.com".

  If the database returns "{^\d(4)}@my.com" the first 4 digits are inserted so the new destination is "1234@my.com" and with "{\d(4)$}@my.com" from the database, the call is sent to "4578@my.com".

6.12 Section [Routing::NeighborSql]

Select which neighbor to query for a call with a database query.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `Query=SELECT ...`
  Default: `N/A`
  Define a SQL query to fetch the new neighbor IP and port.

6.13 Section [Routing::NumberAnalysis]

This section defines rules for the `numberanalysis` routing policy. The policy checks a dialed number for minimum and/or maximum number of digits and sends ARJ, if necessary (number of digits is out of range), to support overlapped digit sending. It also partially supports Setup messages (no overlapped sending - only number length validation).

Format:

`prefix=MIN_DIGITS[:MAX_DIGITS]`

If the number matches the `prefix`, it is verified to consist of at least `MIN_DIGITS` digits and (if MAX_DIGITS is present) at most `MAX_DIGITS` digits. Special wildcard characters (`!`, `'.'` and `'%'`) are available. If the number is too short, an ARJ is send with `rejectReason` set to `incompleteAddress`. If the number is too long, an ARJ is send with `rejectReason` set to `undefinedReason`. Prefix list is searched from the longest to the shortest prefix for a match. For Setup messages, a Release Complete with "badFormatAddress" is sent when the number has an incorrect length.

Example:

```
[RoutingPolicy::OnARQ]
default=numberanalysis,internal

[Routing::NumberAnalysis]
0048=12
48=10
.=6:20
```

Calls to destinations starting with 0048 require at least 12 digits, to 48 we require 10 digits and to all other destinations at least 6 and at most 20 digits.

6.14 Section [Routing::Forwarding]

This routing policy performs a database lookup if calls to an endpoint should be forwarded to another endpoint. It supports routing OnARQ, OnSetup and OnLRQ.

There are different types of forwards:

- **Call Forwarding Unconditional** (CFU, code 1): Calls are always forwarded.
- **Call Forwarding on Busy** (CFB, code 2): Calls are forwarded if the called endpoint is already in a call.
- **Call Forwarding on No Answer** (CFNA, code 3): Calls are forwarded if the called endpoint doesn't answer the call within the AlertingTimeout.
- **Call Forwarding on Error** (CFE, code 4): Calls are forwarded if there is an error routing the call to the endpoint. Currently this behaves like Call Forwarding on No Answer and only one of them should be defined.

The destination where calls are forwarded to should either be aliases of local endpoints (incl. permanent endpoints) or IP numbers. For local aliases, GnuGk will check if the destination also has forwarding configured and take it into account.

Use the 4.3 (common database configuration options) to define your database connection for this module.

Specifically for this module, you can specify a query to read the forwarding rules:

- `Query=SELECT ...`
  Default: `N/A`
  Define a SQL query to fetch the forwarding rules. The query must return 2 colums: First the code for the forwarding type and second the new destination. It must ensure that the results are ordered ascending by forwarding code.

  The query is parameterized - that means parameter replacement is made before each query is executed. The following parameters are defined:

- %c - the called alias
- %p - the called IP (only available on Setup, empty otherwise)
- %s - the calling IP
- %r - the calling aliases
- %{Calling-Station-Id} - the calling station ID (same value as used in accounting and authentication events)
- %i - the call ID
- %m - the message type (ARQ or Setup)
- %{client-auth-id} - a 64 bit integer ID provided to GnuGk when authenticating the call (through SQLAuth)
- %{language} - language if available

Some of these can be empty if they aren't included in the ARQ or Setup message. In most cases you should only use the called alias in the SQL query, since they apply only to the incoming call and won't change when looking up chained forwarding rules.

Example:

```
[RoutedMode]
GKRouted=1
AcceptUnregisteredCalls=1
; failover must be on for forward on timeout
ActivateFailover=1
FailoverCauses=1-15,17-127
DisableRetryChecks=1
; 10 sec alerting timeout (for forward on no answer)
AlertingTimeout=10000

[RoutingPolicy]
default=explicit,forwarding,internal,neighbor,explicit

[Routing::Forwarding]
Driver=MySQL
Host=localhost
Database=gnugk
Username=gnugk
Password=secret
Query=SELECT forwardtype, destination FROM forwards WHERE called = '%c' order by forwardtype asc
MinPoolSize=1
```

Sample MySQL Schema:

```
create table gnugk.forwards (
    called varchar(30) not null,
    forwardtype smallint not null,
    destination varchar(30) not null default "",
    PRIMARY KEY (called, forwardtype)
);
```

Sample Forwarding Rules:

```
"1234", 1, "2000"
"5678", 2, "4000"
"5678", 3, "4000"
"9876", 4, "5000"
```

6.15 Section [Routing::CatchAll]

- `CatchAllIP=1.2.3.4`
  Default: `(empty)`
  Specify an IP address to route all calls to. This overrides CatchAllAlias.
- `CatchAllAlias=Frank`
  Default: `catchall`
  If CatchAllIP is not specified, then route all calls to this alias.

6.16 Section [Routing::Lua]

For details on the LUA language, please see *http://www.lua.org/docs.html* `<http://www.lua.org/docs.html>`.

The LUA script has the following **input variables** available:

- source - source IP
- calledAlias - called alias (only first alias)
- calledIP - called IP address if IP dialing was used
- caller - aliasses of the caller
- callingStationId - calling station ID
- callid - the call ID
- messageType - the message that triggered the routing process ('ARQ', 'LRQ', 'Setup' or 'Facility')
- clientauthid - client auth ID

The LUA script can set these **output variables** to specify a routing destination:

- action - set this to either 'SKIP' or 'REJECT' if you don' want to route the call, otherwise the call is routed to destAlias or destIp (see below)
- rejectCode - reject reason to use with 'REJECT'
- destAlias - call destination alias
- destIP - call destination IP

To access external resources, LUA scripts can use LUA libraries, eg. LuaSocket.

- `Script=destAlias=string.gsub(calledAlias, "#", "*")`
  Default: `(empty)`
  Define the LUA script to run, right in the config file.
- `ScriptFile=script.lua`
  Default: `(empty)`
  Specify a file with a LUA script to run for the 'lua' policy.

6.17 Section [Routing::URIService]

URI Service specific routing policy.

Format:

```
<schema>=<protocol gateway>
```

Example:

```
        [Routing::URIService]
     xmpp=mygateway.mydomain.com
```

This switch sets the service type and default gateway for a given URI schema. This can be used in a chain with [Routing::ENUM::<schema>] and [Routing::SRV::<schema>] to provide a service specific routing policy.

6.18 Section [RewriteCLI]

This section contains a set of rewrite rules for ANI/CLI/H.323_ID numbers (Caller ID). The rewrite process is done in two stages - inbound rewrite and outbound rewrite. The inbound rewrite is done before any other Q.931 Setup message processing (such as inbound GWRewrite, authentication, accounting, ...), and because it alters the Calling-Station-Id it will have an effect in the authorization and accounting modules. The outbound rewrite takes place just before the Setup message is to be forwarded and its effect is visible only to the callee.

An inbound rewrite rule can be matched by a caller's IP and a dialed number or an original CLI/ANI. An outbound rewrite rule can be matched by a caller's IP, callee's IP and a dialed number or a destination number (the dialed number after rewrite) or a CLI/ANI (after inbound rewrite).

This module also provides CLIR (Calling Line Identification Restriction) feature that can be configured for each endpoint (rule).

- `ProcessSourceAddress=1`
  Default: `1`
  In addition to rewriting a Calling-Party-Number Information Element ("IE"), the sourceAddress element of a H.225.0 Setup message can be rewritten, so both contain consistent information.

- `RemoveH323Id=1`
  Default: `1`
  When a sourceInfo element of an H.225.0 Setup message is rewritten, aliases of type H323_ID, email_ID and url_ID can be left untouched if this option is disabled.

- `CLIRPolicy=apply`
  Default: `N/A`
  A global Presentation Indicator ("PI") processing policy can be set up. This policy will be applied to all CLI rewrite rules that do not override it. Possible choices are `forward` - just forward the received PI as-is, `apply` - examine the received PI and hide CLI if it is set to "presentation restricted" and `applyforterminals` - similar to `apply` except that the number is removed only when the call is sent to a terminal, not a gateway.

Format for an inbound rule:

```
in:CALLER_IP=[pi=[allow|restrict][,forward|apply|applyforterminals]]
[cli:|dno:]number_prefix(=|*=|~=|^=|/=)NEW_CLI[,NEW_CLI]...
```

The `in:` prefix specifies that this is an inbound rule and the `CALLER_IP` will be used to match the rule (it can be a single IP or an entire subnet). You can use IPv4 or IPv6 addresses for the `CALLER_IP`.

The optional `pi=` parameter controls CLIR (Calling Line Identification Restriction) features. Specifying either `allow` or `restrict` forces presentation indicator to be set to "presentation allowed" or "presentation restricted". `forward`, `apply` and `applyforterminals` controls how the received (if any) presentation indicator is processed by the gatekeeper. `forward` means forward it to the callee as-is, `apply` is used to hide the CLI if the PI is set to "presentation restricted", `applyforterminals` is similar to `apply`, except that CLI is hidden only when sending the call to a terminal, not a gateway.

The prefix `cli:` or `dno:` (the default) selects what number will be used to match the `number_prefix` - a caller id (CLI/ANI) or a dialed number. Number matching/rewriting can be done in five ways:

- `=` - a `cli` or `dno` number will be matched using a prefix match against `number_prefix` and, if the match is found, CLI will be replaced with NEW_CLI.
- `~=` - a `cli` or `dno` number will be matched using an identity match against `number_prefix` and, if both numbers are the same, CLI will be replaced with NEW_CLI.
- `*=` - (VALID ONLY FOR `cli`) a `cli` number will be matched using a prefix match against `number_prefix` and, if the match is found, the matched CLI prefix (`number_prefix`) will be replaced with a NEW_CLI prefix.
- `^=` - a `cli` or `dno` number will be matched using a prefix match against `number_prefix` and, if the match is found, H.323_ID will be replaced with NEW_CLI, Calling-Station-Id will remain unchanged.
- `/=` - a `cli` or `dno` number will be matched using an identity match against `number_prefix` and, if both numbers are the same, H.323_ID will be replaced with NEW_CLI, Calling-Station=Id will remain unchanged,

After the equality (=/ =/*=/ˆ=//=) sign, there follows a list of new CLI values to be used. If more than one value is specified, one will be chosen on a random basis. It's possible to specify whole number ranges, like 49173600000-49173699999 (for number ranges CLIs should have a fixed length). There is a special string constant "any" which may be used in place of the `CALLER_IP` or the `number_prefix`. To enable `CLIR` for this rule, use the special string constant `"hide"` instead of the list of new CLI values. Note that CLIR is far more useful for outbound rules.

Example 1:

```
[RewriteCLI]
in:192.168.1.1=dno:5551=3003
in:192.168.1.1=cli:1001=2222
in:192.168.1.1=any=1111
```

These rules state that for calls from the IP 192.168.1.1: 1) if the user dialed a number beginning with 5551, set CLI to 3003, 2) if the call is from user with CLI beginning with 1001, set CLI to 2222, 3) for other calls from this IP, set CLI to 1111.

Example 2:

```
[RewriteCLI]
in:192.168.1.0/24=any=18001111
in:192.168.2.0/24=any=18002222
in:2002:4ad0:ff00:79a::2/64=any=18003333
in:any=any=0
```

These rules state that: 1) for calls from the network 192.168.1.0/24, set CLI to 18001111, 2) for calls from the network 192.168.2.0/24, set CLI to 18002222, 3) for calls from the network 2002:4ad0:ff00:79a::2/64, set CLI to 18003333, 4) for other calls, set CLI to 0.

Example 3:

```
[RewriteCLI]
in:192.168.1.0/24=0048*=48
in:192.168.1.0/24=0*=48
in:any=100.~=48900900900
```

These rules state that: 1) for calls from the network 192.168.1.0/24, rewrite 0048 to 48 (example - 0048900900900 => 48900900900), 2) for other calls from the network 192.168.1.0/24, rewrite 0 to 48 (example - 0900900900 => 48900900900), 3) for other calls, if CLI is 4 digits and starts with 100, set it to 48900900900.

Example 4 (CLIR):

```
[RewriteCLI]
in:192.168.1.0/24=any=hide
```

This example causes caller's number to be removed from Setup messages originating from the 192.168.1.0/24 network. It also causes proper presentation and screening indicators to be set in Setup messages.

Format for an outbound rule:

```
out:CALLER_IP=CALLEE_IP [pi=[allow|restrict][,forward|apply|applyforterminals]]
[cli:|dno:|cno:]number_prefix(=|~=|*=)NEW_CLI[,NEW_CLI]...
```

The `out:` prefix tells that this is an outbound rule, the `CALLER_IP` and the `CALLEE_IP` will be used to match the rule and can be a single IP or a subnet address.

The optional `pi=` parameter controls CLIR (Calling Line Identification Restriction) features. Specifying either `allow` or `restrict` forces the presentation indicator to be set to "presentation allowed" or "presentation restricted". `forward`, `apply` and `applyforterminals` controls how the received (if any) presentation indicator is processed by the gatekeeper. `forward` means just to forward it to the callee as-is, `apply` means hiding CLI if the PI is set to "presentation restricted", `applyforterminals` is similar to `apply`, except that the CLI is hidden only when sending the call to a terminal, not a gateway.

The prefix `cli:`, `dno:` (the default) or `cno:` selects what number will be used to match the `number_prefix` - a caller id (CLI/ANI), a dialed number or a destination/called number (the dialed number after rewrite). Number matching/rewriting can be done in three ways:

- `=` - a `cli` or `dno` number will be matched using a prefix match against `number_prefix` and, if the match is found, CLI will be replaced with NEW_CLI,
- `~=` - a `cli` or `dno` number will be matched using an identity match against `number_prefix` and, if both numbers are the same, CLI will be replaced with NEW_CLI,
- `*=` - (VALID ONLY FOR `cli`) a `cli` number will be matched using a prefix match against `number_prefix` and, if the match is found, the matched CLI prefix (`number_prefix`) will be replaced with a NEW_CLI prefix.

After the equality sign (=/ =/*=), a list of new CLI values to be used is specified. If more than one value is configured, one will be chosen on a random basis. It's possible to specify entire number ranges, like 49173600000-49173699999. There is a special string constant "any" which can be used in place of the `CALLER_IP`, the `CALLEE_IP` or the `number_prefix`. To enable `CLIR` for this rule, use a special string constant `"hide"` or `"hidefromterminals"` instead of the list of new CLI values.

Example 1:

```
[RewriteCLI]
out:any=192.168.1.1 any=1001
out:any=192.168.1.2 any=1002
out:any=any cno:123=1003
```

These rules set a fixed ANI/CLI for each terminating IP: 1) present myself with ANI 1001, when sending calls to IP 192.168.1.1, 2) present myself with ANI 1002, when sending calls to IP 192.168.1.2. 3) present myself with ANI 1003, when calling 123

Example 2:

```
[RewriteCLI]
out:any=192.168.1.1 any=1001-1999,3001-3999
```

This rule randomly selects ANI/CLI from range 1001-1999, 3001-3999 for calls sent to 192.168.1.1.

Example 3 (CLIR):

```
[RewriteCLI]
out:any=any any=hidefromterminals
out:192.168.1.1=any any=hide
```

In this example each subscriber has enabled CLIR, so all calls to terminals will have a caller's number removed and presentation/screening indicators set. Calls to gateways will have the presentation indicator set to "presentation restricted" and the caller's number will not be removed to allow proper call routing and number removal at the destination equipment.
One exception to these rules are calls from 192.168.1.1 which will have a caller's number always removed, no matter whether calling a terminal or a gateway.

Example 4 (CLIP):

```
[RewriteCLI]
out:any=192.168.1.1 any=hide
```

In this example CLIP (Calling Line Identification Presentation) feature is disabled for the user 192.168.1.1.

Example 5 (CLIR):

```
[RewriteCLI]
out:192.168.1.1=any pi=restrict,apply cli:.*=.
out:any=any pi=allow cli:.*=.
```

These rules do not change CLI (.*=.) and: 1) enable CLIR for an endpoint 192.168.1.1. `apply` tells the gatekeeper to not only set the PI, but also to hide the number. 2) force CLI presentation for other endpoints.

The rule matching process has a strictly defined order:

1. the closest caller's IP match is determined (closest means with the longest network mask - single IPs have the highest priority, "any" has the lowest priority),

2. (outbound rules) the closest callee's IP match is determined,
3. the longest matching prefix/number is searched for the given IP/IP pair in the following order:
   (a) `dno:` type (dialed number) rules are searched,
   (b) `cno:` type (destination/called number) rules are searched,
   (c) `cli:` type (caller id) rules are searched.

After a match for caller's/caller's IP is found, no more rules are checked, even if no prefix/number is matched inside the set of rules for these IPs.

On the Windows platform, there is a problem with duplicated config keys in INI files, so GnuGk provides a workaround for this restriction. This example will not work because of the same key (`in:192.168.1.1`):

```
[RewriteCLI]
in:192.168.1.1=1001=2001
in:192.168.1.1=any=2000
```

As a workaround, you can use a string with percent signs (%) at the beginning and at the end before the key. This prefix will be automatically stripped from the key name before loading rules:

```
[RewriteCLI]
%r1% in:192.168.1.1=1001=2001
%r2% in:192.168.1.1=any=2000
```

6.19 Section [RewriteCLI::SQL]

Use the 4.3 (common database configuration options) to define your database connection for this module.

Please note that the switches (not the rules) from the 6.18 (RewriteCLI) section, like `ProcessSourceAddress=`, `RemoveH323Id=` and `CLIRPolicy=` also apply to the rewrite rules from this section.

- `InboundQuery=SELECT ...`
  Default: `N/A`
  Define a rewriting query to run when the call comes in.
- `outboundQuery=SELECT ...`
  Default: `N/A`
  Define a rewriting query to run when the call is sent out. The called number parameter has already passed all rewriting steps.

The first field returned by the query is used as the new CLI. If the query returns no rows, the CLI is left unchanged. The queries can be parameterized - that means parameter replacement is made before each query is executed. The following parameters are defined:

- `%{cli}` - the original CLI or first sourceAddress if no CLI exists (on outbound queries, it can already be rewritten by an Inbound query)
- `%{callerip}` - the calling IP
- `%{called}` - the called number (the dialed number on inbound queries and the rewritten number in outbound queries)

In most cases you will probably only use the %{cli} parameter.

6.20 Section [RewriteSourceAddress]

With the switches in this section you can filter the sourceAddress elements that are transported in a Setup message. (Please note that the 6.18 (RewriteCLI) and 6.19 (RewriteCLI::SQL) rules also influence the sourceAddress.)

- `OnlyE164=1`
  Default: `0`
  With this switch you can filter out all elements that are not of type E.164.
- `OnlyValid10Dand11D=1`
  Default: `0`
  With this switch you can filter out all elements that are not valid 10-digit or 11-digit US numbers. They may be of any alias type (unless OnlyE164 is set), but no formatting characters are allowed. 11-digit numbers must start with 1 and area codes must start with 2..9.
- `MatchSourceTypeToDestination=1`
  Default: `0`
  With this switch you can filter out all elements that do not match the destination Type (E.164 or URI) If you call an E.164 number (Q931 IE: Called-Party-Number present) everything other then dialdigit source will be filtered. If you call a URI (destination AliasAddress type) everything other then URI source will be filtered. This switch has no effect on any other destination type. For example H323ID or TransportID AliasTypes.
- `ForceAliasType=1`
  Default: `-1`
  values 0-dialedDigits 1-h323_ID 2-URI-ID With this switch you can force the source and destination AliasAddress to the supplied type. Used in conjunction with MatchSourceTypeToDestination to change the AliasType. for instance change url_ID to h323_ID so the remote gateway can process the message.
- `ReplaceChar=+,0;#,*`
  Default: `N/A`
  With this switch you can remove/replace characters on the callers source address such as +.
- `Rules=01,18001234567`
  Default: `N/A`
  With this switch you can replace the CallSourceAddress if there is a prefix match. You can use this to assign a common valid E.164 number to non-E.164 numbers for the purpose of callerID.
- `TreatNumberURIDialedDigits=1`
  Default: `0`
  Where MatchSourceTypeToDestination is set and the destination is a URI and the host part is numeric this ensures the source address is numeric URI as well by taking the DialedDigits source address and mixing it with the URI address if present so the same format is for the source and destination address.

7 RAS Configuration

7.1 Section [ReplyToRasAddress]

Some messages from endpoints to the gatekeeper (GatekeeperRequest, RegistrationRequest and InfoRequestResponse) contain an element *rasAddress* where the endpoint tells the gatekeeper where to send the response to these messages. By default GnuGk will ignore this address and respond to the IP and port where it has received the request from. It does so, because some endpoints rely on this behavior and in case where eg. a NAT is used, the response may not reach the sender if it is sent to another IP and port. Usually endpoints will send the RAS messages from their RAS port anyway, so it doesn't make a difference.

This section allows you to define when GnuGk should use the rasAddress inside the message it has received instead of the address where it has received the message from.

Syntax:

```
network=True|False
```

The network is specified by an IP plus optional CIDR, eg. 192.168.1.0/24. The network specifies the IP where the RAS message is received from, the setting specifies whether to use the rasAddress. The default is not to use it. The rule for the network with the longest netmask is used (the most specific).

Example:

In this example messages from the 192.168.0.0/18 will use the rasAddress, except for messages coming from the 192.168.4.0/24 network.

```
[ReplyToRasAddress]
192.168.0.0/18=True
192.168.4.0/24=False
```

7.2 Section [RasSrv::GWPrefixes]

This section configures how dialed E.164 numbers are routed to a specific gateway.

Format:

```
gw-alias=prefix[:=priority][,prefix[:=priority],...]
```

Note that you must specify the alias of the gateway. If a gateway has registered with the specified alias, all numbers beginning with the prefixes are routed to that gateway. Special characters `.` and `!` can be used here to match any digit or to disable the prefix. A priority can be given to each prefix for each gateway (using := syntax), so that if several gateways match the dialed number, the one with the highest prefix priority will be selected to route the call (when the ActivateFailover switch is ON, the call will be routed to all selected gateways in order of the prefix priority). A smaller value corresponds to a higher priority. Default value is 1. If the prefix priority and overlaps the GatewayPriority (see section 11 ([EP::...])), the prefix priority will be preferred.

In the following example, the gateway "test-gw" will be responsible for prefixes "02" and "03" with a priority of 3, and for "04" with a priority of 1.

Example:

```
test-gw=02,03:=3,04:=1
```

7.3 Section [RasSrv::PermanentEndpoints]

In this section you may configure endpoints that don't have RAS support or that you don't want to be expired. Their records will always remain in the registration table of the gatekeeper. However, you can still unregister it via the status port. Special characters `.` and `!` can be used with prefixes here to match any digit and disable the prefix. You may use := syntax to set a prefix priority in the same manner as in 7.2 ([RasSrv::GWPrefixes]) section.

Make sure you add at least one prefix for all gateways, even if you assign the prefixes elsewhere (eg. in the [EP::...] section), otherwise the endpoint won't be considered a gateway and those settings won't apply!

Gateway entries may also optionally include vendor information which is stored with the gateway record

Format:

```
IP[:port]=alias[,alias,...;prefix[:=priority][,prefix[:=priority]]...;[vendor,product]
```

Example:

For gateway,

```
10.0.1.5=MyGW;009,008:=2,0.7:=3
```

```
10.0.1.5=MyGW;009,008:=2,0.7:=3;yate,4.1.0
```

For terminal,

```
10.0.1.10:1720=700
```

7.4 Section [RasSrv::RRQFeatures]

- `AcceptEndpointIdentifier=1`
  Default: `1`
  Whether to accept **endpointIdentifier** specified in a full RRQ.

- `AcceptGatewayPrefixes=1`
  Default: `1`
  A gateway can register its prefixes with the gatekeeper by sending **supportedPrefixes** in the **terminalType** field of the RRQ. This option defines whether to accept the specified prefixes of a gateway.

- `AcceptMCUPrefixes=1`
  Default: `1`
  A MCU can register its prefixes with the gatekeeper by sending **supportedPrefixes** in the **terminalType** field of the RRQ. This option defines whether to accept the specified prefixes of a MCU.

- `OverwriteEPOnSameAddress=1`
  Default: `0`
  In some networks an endpoint's IP address may change unexpectedly. This may happen when an endpoint is using a PPP connection (e.g. modem or ADSL). This option defines how to handle a registration request (RRQ) from an IP address which does not match what we have stored. The default action is to reject the request. With this option enabled the conflicting request will cause an unregister request (URQ) to be sent for the existing IP address and the entry to be removed, allowing the endpoint to register with the new address.

- `IRQPollCount=0`
  Default: `1`
  When the gatekeeper does not receive a keep-alive RRQ from an endpoint within the TimeToLive time period, it sends an IRQ message to "poll" the endpoint and check if it is alive. After IRQPollCount messages are sent and no reply is received, the endpoint is unregistered. To disable this feature (and unregister endpoints immediately after TimeToLive timeout), set this variable to 0. IRQ poll interval is 60 seconds.

- `SupportDynamicIP=1`
  Default: `0`
  When the IP address of an endpoint changes, the gatekeeper can maintain registration. This will force the EP to fully re-register if its IP address changes.

- `AccHTTPLink=https://billing.mysite.com?account=%a&password=%p`
  Default: `N/A`
  You can assign a URL for clients to access to view billing information. If using PacPhone you can also add wildcards for the client to use so the clients H323ID and password can be used to directly access their account information. %a - H323ID %p - password

- `AliasTypeFilter=terminal;h323id,dialeddigits`
  Default: `N/A`
  Use this setting where endpoints send multiple H225_AliasAddress and some Aliases are shared across multiple registrations. You can filter out the shared alias types for any given endpoint type. The registrations will keep all alias types listed in the filter setting and remove all others. You must have separate AliasTypeFilter entries for each endpoint type. Valid endpoint types are: gatekeeper, gateway, mcu and terminal. Valid filters are: h323id, dialeddigits, url, transport, email and partynumber. NOTE: If no alias is found that match the filter then all aliases are registered.

- `GatewayAssignAliases=0`
  Default: `1`
  If AssignedAliases::SQL has been configured, apply assignments to gateway registrations (default). This switch is designed when set to 0 to use additiveRegistrations with gateways so that the assignedAliases are not all assigned upon registration but only when the additiveRegistration is made. This ensures only the currently registered endpoints appear in the endpoint table.

- `AuthenticatedAliasesOnly=1`
  Default: `0`
  This switch removes during registration any TerminalAliases that have not been Authenticated. Supported Authentication modules are SQLAuth, SimplePasswordAuth and SQLPasswordAuth.

7.5 Section [RasSrv::ARQFeatures]

- `ArjReasonRouteCallToGatekeeper=1`
  Default: `1`
  If enabled, the gatekeeper rejects an answered ARQ without a pre-existing CallRec found in the CallTable by reason **routeCallToGatekeeper** in routed mode. The endpoint shall release the call immediately and re-send call Setup to the gatekeeper.

- `RemoveTrailingChar=#`
  Default: `N/A`
  Specify the trailing character to be removed in **destinationInfo**. For example, if your endpoint incorrectly contains a termination character such as '#' in **destinationInfo** you may remove it with this option.

- `RoundRobinGateways=0`
  Default: `1`
  Enable/disable round-robin gateway selection if more than one gateway matches a dialed number. If disabled, the first available gateway will be selected. Otherwise, subsequent calls will be sent to gateways in round-robin fashion.
- `SendRIP=9000`
  Default: `0`
  Send a RequestInProgress (RIP) message with this delay value after receiving an ARQ. This switch can be used to extend the duration the caller will wait for an answer. No RIP is sent when the delay ist set to 0.
- `CheckSenderIP=1`
  Default: `0`
  Verify that the ARQ is sent from the same IP as the RRQ.

7.6 Section [RasSrv::AssignedAlias]

This allows the assigning of aliases to endpoints as they register, allowing them to set their fully qualified E.164 or URI addresses.

Example:

```
[RasSrv::AssignedAlias]
1234=3323465777,me@mysite.com
```

7.7 Section [AssignedAliases::SQL]

This section configures GnuGk to read the assigned aliases from a database. You can use the same database parameters as defined in 8.4 ([SQLPasswordAuth]).

- `Query=SELECT ...`
  Default: `N/A`
  Defines the SQL query used to retrieve the assigned aliases from the database.

  One parameter is defined:

  - `%u` - endpoint alias

  Sample query string:

  ```
  SELECT assignedalias FROM users WHERE alias = '%u' AND active
  ```

7.8 Section [RasSrv::AlternateGatekeeper]

This section allows you to override the global definition of 4.5 (AlternateGKs) from the [Gatekeeper::Main] section for certain IPs or IP ranges. See there for a detailed definition of the config options.

The network is specified by an IP plus optional CIDR, eg. 192.168.1.0/24. The rule for the network with the longest netmask is used (the most specific).

Example:

In this example, 192.168.1.10 gets assigned GnuGk10 as alternate gatekeeper, while the rest of the 192.168.0.0/18 network will use GnuGk4. Endpoints in all other networks will use the globally defined alternate gatekeeper.

```
[RasSrv::AlternateGatekeeper]
192.168.0.0/18=1.2.3.4;1719;true;120;GnuGk4
192.168.1.10=1.2.3.10;1719;true;120;GnuGk10
```

7.9 Section [RasSrv::AssignedGatekeeper]

This allows the assigning of a gatekeeper based upon the H323ID or the apparent source IP address of the registering endpoint. The received H323ID in the GRQ is checked to see if it has a prefix for an assigned gatekeeper or the IP is in a range of an assigned gatekeeper. The endpoint is then advised in the GCF to register with that gatekeeper. You may have multiple gatekeepers for a specific prefix. The first is assigned as the primary and others are then the alternates. (requires H.323v6)

Examples:

```
[RasSrv::AssignedGatekeeper]
;; for endpoint with alias starting with 01234
01234=192.168.1.100:1719
;; for endpoint with alias starting with 999
999=[2a01:4f8:61:2243::99]:1719
;; for endpoints in the range of 195.71.129.0/24 or 195.71.131.0/24
^195\.71\.(129|131)\.[0-9]+$=10.10.0.5:1719
;; for endpoints tarting with ^2a01:
^2a01:=[2a01:4f8:61:2243::199]:1719
```

7.10 Section [AssignedGatekeepers::SQL]

This section allows *GnuGk* to read the assigned gatekeepers from a database. You can use the same database parameters as defined in 8.4 ([SQLPasswordAuth]).

- `Query=SELECT ...`
  Default: `N/A`
  Defines the SQL query used to retrieve the assigned gatekeepers from the database.

  These parameters are defined:

  – `%u` - endpoint alias
  – `%i` - endpoint IP
  – `%g` - gatekeeper ID

  Sample query string:

  ```
  SELECT assignedgatekeeper FROM users WHERE alias = '%u' AND active
  ```

7.11 Section [AlternateGatekeepers::SQL]

This section allows *GnuGk* to read the alternate gatekeepers from a database. You can use the same database parameters as defined in 8.4 ([SQLPasswordAuth]).

- `Query=SELECT ...`
  Default: `N/A`
  Defines the SQL query used to retrieve the alternate gatekeepers from the database.

  One parameter is defined:

  - `%i` - endpoint IP

  Sample query string:

  ```
  SELECT alternategatekeeper FROM users WHERE ip = '%i' AND active
  ```

7.12 Section [AssignedLanguage::SQL]

This section configures GnuGk to read the assigned Languages from a database. You can use the same database parameters as defined in 8.4 ([SQLPasswordAuth]).

- `Query=SELECT ...`
  Default: `N/A`
  Defines the SQL query used to retrieve the assigned languages from the database.

  One parameter is defined:

  - `%u` - endpoint alias

  Sample query string:

  ```
  SELECT assignedlanguage FROM users WHERE alias = '%u' AND active
  ```

7.13 Section [NATedEndpoints]

The gatekeeper can automatically detect whether an endpoint is behind NAT. However, if the detection fails, you can specify it manually in this section.

Format:

```
alias=true | yes | 1
```

Example:

Specify that the endpoint with alias 601 is behind NAT.

```
601=true
```

7.14 Section [GkPresence::SQL]

H323 SQL Presence system : Highly Experimental Use the 4.3 (common database configuration options) to define your database connection for this module.

- `IncrementalUpdate=1`
  Default: `0`
  Whether to poll the database to check for Presence updates

- `UpdateWorkerTimer=10`
  Default: `5`
  Sleep time between updating of H.460 Presence information.

- `QueryList=SELECT ...`
  Default: `N/A`
  Define a SQL query to be used to perform database lookups to retreive contact information (order is important)

  - `%i` - presence identifier GUID value create externally.
  - `%u` - alias to which the presence information belongs
  - `%a` - contact alias
  - `%s` - Is subscriber (default should be 0)
  - `%b` - contact instruction values 0-subscribe 1-unsubscribe 2-block 3-unblock 4-waiting approval
  - `%y` - Whether instruction active (default should be 1)
  - `%z` - Update Time (should be current UNIX time)
  - `%d` - Display name or friendly name of the alias (optional)
  - `%v` - Path to the contacts URL (optional)
  - `%c` - contact category (optional) values 0-Audio 1-Video 2-data 3-H.239 4-generic
  - `%t` - Incremental Timestamp for query (set by Gatekeeper)

  Example QueryList=SELECT subscriptionID,h323id,alias,0,status,1,updated,display,avatar,category FROM subscription WHERE timestamp > '%t' ORDER BY h323id,alias

- `QueryAdd=INSERT ...`
  Default: `N/A`
  Define a SQL query to Add contact record to the database Example QueryAdd=INSERT INTO subscription (subscriptionID,h323id,alias,isSubscriber,display) VALUES('%i','%u','%a','%s', '%d');

- `QueryDelete=Delete ...`
  Default: `N/A`
  Define a SQL query to delete a contact from the database Example QueryDelete=DELETE FROM subscription WHERE subscriptionID = '%i'

  item>`QueryUpdate=UPDATE ...`
  Default: `N/A`

  Define a SQL query to Update contact record status Example QueryUpdate=UPDATE subscription SET status = '%b' WHERE subscriptionID = '%i'

8 Authentication Configuration

The following sections in the config file can be used to configure authentication.

8.1 Section [Gatekeeper::Auth]

The section defines the authentication mechanism / modules for the *GNU Gatekeeper*. After defining which modules to use, you have to add the corresponding config sections for each module.

Syntax:

```
authrule=actions

 <authrule> := SimplePasswordAuth | AliasAuth | FileIPAuth | PrefixAuth | RadAuth | RadAliasAuth | SQL
 <actions>  := <control>[;<ras>|<q931>,<ras>|<q931>,...]
 <control>  := optional | required | sufficient | alternative
 <ras>      := GRQ | RRQ | URQ | ARQ | BRQ | DRQ | LRQ | IRQ
 <q931>     := Setup | SetupUnreg
```

A rule may result in one of three codes: ok, fail, next.

- `ok` - The request is authenticated by this module.
- `fail` - The authentication fails and should be rejected.
- `next` - The rule cannot determine the request.

There are also three ways to control a rule:

- `optional` - If the rule cannot determine the request or accepts it, it is passed to next rule. Otherwise processing stops and the request is rejected.
- `required` - The requests should be authenticated by this module, or it would be rejected. The authenticated request would then be passed to next rule.
- `sufficient` - If the request is authenticated, it is accepted, or it would be rejected. That is, the rule determines the fate of the request. No rule should be put after a sufficient rule, since it won't take effect.
- `alternative` - similar to the `sufficient` rule, except that if the module cannot determine the result the request is passed to a next module.

Currently supported modules: (most only support a subset of the RAS or Q.931 actions)

- `SimplePasswordAuth/SQLPasswordAuth/H350PasswordAuth` These modules check the tokens or cryptoTokens fields of a RAS message. The tokens should contain at least generalID and password. For cryptoTokens, cryptoEPPwdHash tokens hashed by simple MD5 and nestedcryptoToken tokens hashed by HMAC-SHA1-96 are supported. For tokens hashed by CAT (Cisco Access Token) a clear text username/password are supported. The ID and password are read from the 8.3 ([SimplePasswordAuth]) section, or a SQL database for `SimplePasswordAuth` and `SQLPasswordAuth` modules. For H.350.2 authentication (`H350PasswordAuth`) the 8.13 ([GkH350::Settings]) connection information must be completed. The 8.5 ([H350PasswordAuth]) section is optional.

- `AliasAuth/SQLAliasAuth` The module can only be used to authenticate RegistrationRequest (RRQ). The IP of an endpoint with a given alias should match a specified pattern. For `AliasAuth` the pattern is defined in the 8.6 ([RasSrv::RRQAuth]) section. For `SQLAliasAuth`, the pattern is retrieved from a SQL database as defined in the 8.7 ([SQLAliasAuth]) section.
- `FileIPAuth` This module provides a simple way to restrict access to the gatekeeper based on caller's IP or network.
- `PrefixAuth` The IP or aliases of a request with a given prefix must match a specified pattern. See section 8.9 ([PrefixAuth]) for details. The module can authorize AdmissionRequest (ARQ), Location-Request (LRQ), Setup and SetupUnreg.
- `RadAuth` Provides authentication based on H.235 username/password security scheme. Authenticates RRQ, ARQ and Q.931 Setup through remote RADIUS servers. It sends usernames and passwords extracted from CAT (Cisco Access Tokens) tokens carried inside RRQ, ARQ or Setup packets to the RADIUS servers. If your endpoints do not support CATs or you do not need an authentication scheme based on individually assigned usernames/password then this module would not be appropriate (but you may check the `RadAliasAuth` module). See section 8.10 ([RadAuth]) for details.
- `RadAliasAuth` Provides authentication based on endpoint aliases and/or call signaling IP addresses with remote RADIUS servers. It does not need any H.235 tokens inside RAS messages, so it can be used on a wider range of systems as compared to `RadAuth`. RRQ, ARQ and Q.931 Setup messages can be authenticated using this module. See section 8.11 ([RadAliasAuth]) for details.
- `SQLAuth` A powerful module to authenticate and authorize RRQ, ARQ, LRQ and Setup messages. It can perform checks based on various parameters such as caller's number, destination number, username and more. It also supports enforcing call duration limit, number rewriting, call routing, alias verification and assignment. See section 8.8 ([SQLAuth]) for more details.
- `LDAPPasswordAuth` This module will lookup the called aliases in your LDAP schema searching the H323ID and telephoneNumber attribute and checks if the password in the H235Password attribute matches the H.235 crypto token.

  The LDAP server is configured in the 8.14 (GkLDAP::Settings) section and the attribute matching is defined in the 8.15 (GkLDAP::LDAPAttributeNames) section.
- `LDAPAliasAuth` This module will lookup the called aliases in your LDAP schema searching the H323ID and telephoneNumber attribute and checks if the IP matches the IPAddress attribute.

  The LDAP server is configured in the 8.14 (GkLDAP::Settings) section and the attribute matching is defined in the 8.15 (GkLDAP::LDAPAttributeNames) section.
- `LuaAuth` This module utilizes user-defined LUA scripts to perform authentication. The scripts are configured in the 8.16 (LuaAuth) section.
- `GeoIPAuth` This module checks the which country the message comes from by looking at the IP. The module is configured in the 8.17 (GeoIPAuth) section.
- `CapacityControl` A flexible module to control inbound call volume with the ability to configure various conditions. IMPORTANT: It must be used in conjunction with the `CapacityControl` accounting module. See section 8.12 ([CapacityControl]) for more details.

You can also configure a rule to check only for specific RAS messages. The following example configures `SimplePasswordAuth` as an optional rule to check RRQ and ARQ. If a RRQ is not checked (does not contain tokens or cryptoTokens fields), it is checked by `AliasAuth`. The default is to check all supported requests.

Example 1:

```
SimplePasswordAuth=alternative;RRQ,ARQ
AliasAuth=sufficient;RRQ
```

The example below authenticates all calls, checking signaling Setup message details, using the RadAliasAuth module.

Example 2:

```
RadAliasAuth=required;Setup
default=allow
```

This example checks endpoint registrations (RRQ) and call admissions (ARQ) either by means of username/password (RadAuth) or alias/IP (RadAliasAuth). Additionally, if the call is from an unregistered endpoint (and therefore no RRQ or ARQ authentication has been performed), Setup message authentication using RadAliasAuth takes place (SetupUnreg).

Example 3:

```
RadAuth=alternative;RRQ,ARQ
RadAliasAuth=alternative;RRQ,ARQ,SetupUnreg
default=reject
```

8.2 Section [FileIPAuth]

This section defines a list of IP addresses/networks which are allowed to access gatekeeper resources. A list of allowed prefixes can be specified together with an IP address. Supported Gatekeeper::Auth events are: `GRQ`, `RRQ`, `ARQ`, `LRQ`, `Setup` and `SetupUnreg`. Format of a single entry is:

`IP=[allow | reject | onlyTLS][;prefix[,prefix...]]`

where IP is a single IP address, a network address (in A.B.C.D/M.M.M.M or A.B.C.D/LENGTH format or IPv6 format) or a string `'any'` or `'*'` to match any address. The access list can also be loaded from an external file using `include` directive. During authentication, network mask length defines a priority for each entry, so rule 192.168.1.1=allow takes precedence over 192.168.1.0/24=reject.

`'onlyTLS'` is equivalent to `'allow'` if the call comes in via a 12.11 (TLS) secured connection and means `'reject'` for unencrypted calls.

In addition the to endpoint's IP, you can specify a list of prefixes that the endpoint may call. The destination prefixes are only checked on ARQ and Setup messages.

Example #1:

```
[Gatekeeper::Auth]
FileIPAuth=required;RRQ,ARQ,LRQ,Setup

[FileIPAuth]
192.168.1.240=reject
192.168.1.0/24=allow
192.168.2.0/255.255.255.0=allow;48,49,44
2a01:4f8:61:2243::2=allow
2a01:4f8:61:2243::10/128=allow
2a01:4f8:61:2243::/64=allow
any=reject
```

Example #2:

Placing the list of IP rules into another file.

```
[Gatekeeper::Auth]
FileIPAuth=required;Setup

[FileIPAuth]
include=/etc/gnugk/accesslist.ini

(EOF)

Contents of /etc/gnugk/accesslist.ini:

[FileIPAuth]
192.168.1.1=allow
192.168.1.100=allow
any=reject
```

Example #3:

Allow all connects from the local network, but require TLS encryption and authentification for everything else.

```
[Gatekeeper::Auth]
FileIPAuth=required;Setup

[FileIPAuth]
192.168.1.0/24=allow
any=onlyTLS
```

8.3 Section [SimplePasswordAuth]

This section defines the userid and password pairs used by `SimplePasswordAuth` module. All passwords are encrypted using the `addpasswd` utility.

Usage:

```
addpasswd config section userid password
```

Example:

```
addpasswd config.ini SimplePasswordAuth frank secret
```

Options:

- `KeyFilled=123`
  Default: `0`
  Default value to use as a padding byte during password encryption/decryption.
- `CheckID=1`
  Default: `0`
  Check if the aliases match the ID in the tokens.

- `PasswordTimeout=120`
  Default: `-1`
  The module `SimplePasswordAuth` will cache an authenticated password. This field defines the cache timeout value in seconds. `0` means never cache the password, while a negative value means the cache never expires.
- `DisableAlgorithm=MD5,H.235.1,CAT`
  Default: `N/A`
  Disable H.235 authentication algorithms in the GRQ/GCF negotiation, otherwise all algorithms supported by GnuGk are used. A disabled algorithm will still be used if it is used by an endpoint without negotiation. This switch can be used to avoid incompatibilities with vendor implementations.

8.4 Section [SQLPasswordAuth]

Authenticate H.235 enabled endpoints using passwords stored in a SQL database. This section defines the SQL driver to use, SQL database connection parameters and the query to use to retrieve passwords.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `CacheTimeout=120`
  Default: `0`
  This field defines how long (alias;password) pairs retrieved from the database will be cached. The cache timeout value is expressed in seconds. `0` means to not cache passwords, while a negative value means the cache never expires (only `reload` command will refresh the cache).
- `Query=SELECT ...`
  Default: `N/A`
  Defines SQL query used to retrieve H.235 password from the database. The query is parameterized - that means parameter replacement is made before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `SQLPasswordAuth` two parameters are defined:
  - `%1` - the actual alias to query the password for
  - `%2` - the gatekeeper identifier

  Sample query strings:

  ```
  SELECT h235password FROM users WHERE alias = '%1' AND active
  SELECT h235password FROM users WHERE alias = '%1' AND gk = '%2'
  ```

8.5 Section [H350PasswordAuth]

- `PasswordTimeout=120`
  Default: `-1`
  The module `SimplePasswordAuth` will cache an authenticated password. This field defines the cache timeout value in seconds. `0` means never cache the password, while a negative value means the cache never expires.

8.6 Section [RasSrv::RRQAuth]

Specify the action on RRQ reception (confirm or deny) for `AliasAuth` module. The first alias (this will mostly be an H323ID) of the endpoint to register is looked up in this section. If a parameter is found the value will apply as a rule. A rule consists of conditions separated by "&". A registration is accepted when all conditions apply. If an endpoint is using H.460.18, the port in the authentication rule will be ignored.

Syntax:

```
<authrules> :=  empty  |  <authrule> "&" <authrules>

  <authrule>  := <authtype> ":" <authparams>
  <authtype>  := "sigaddr" | "sigip"
  <autparams> := [!&]*
```

The notation and meaning of <`authparams`> depends on <`authtype`>:

- `sigaddr` - extended regular expression that has to match against the "PrintOn(ostream)" representation of the signal address of the request. Example:

  ```
  sigaddr:.*ipAddress .* ip = .* c0 a8 e2 a5 .*port = 1720.*
  ```

- `sigip` - specialized form of '`sigaddr`'. Write the signaling IP address. If the port is omitted, 1720 is assumed.

  Examples:

  ```
  sigip:192.168.242.165:1720
  sigip:[2a01:4f8:61:2243::2]:1720
  sigip:2a01:4f8:61:2243::2
  ```

- `allow` - always accept the alias.
- `deny` - always reject the alias.

Example:

```
[RasSrv::RRQAuth]
; The endpoint with alias 'cwhuang' must register from 10.0.1.10:1720
cwhuang=sigip:10.0.1.10:1720
; The endpoint with alias 'gw1' must register from 10.0.1.0/24
gw1=sigaddr:.*ipAddress .* ip = .* 0a 00 01 .*port = 1720.*
; The endpoint with alias 'gw2' must register from [2a01:4f8:61:2243::2]:1720
gw2=sigip:[2a01:4f8:61:2243::2]:1720
```

8.7 Section [SQLAliasAuth]

Authenticate endpoints using rules stored in the SQL database (the rules conform to the format defined in the 8.6 ([RasSrv::RRQAuth]) section). This section defines which SQL driver to use, SQL database connection parameters and the query to use to retrieve the patterns.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `CacheTimeout=120`
  Default: `0`
  This field defines how long (alias;authrule) pairs retrieved from the database will be cached. The cache timeout value is expressed in seconds. `0` means not to cache rules, while a negative value means the cache never expires (only `reload` command will refresh the cache).

- `Query=SELECT ...`
  Default: `N/A`
  Defines the SQL query used to retrieve alias rule from the database. The query is parameterized - that means parameter replacement is made before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `SQLAliasAuth` two parameters are defined:

  - `%1` - the actual alias to query the rule for
  - `%2` - the gatekeeper identifier

  Sample query strings:

  ```
  SELECT authrule FROM users WHERE alias = '%1' AND active
  SELECT 'sigip:' || host(ip) || port FROM users WHERE alias = '%1'
  ```

8.8 Section [SQLAuth]

Authenticate and authorize endpoints/calls using a SQL database. Support for RRQ, ARQ, LRQ and Setup events is provided.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `RegQuery=SELECT ...`
  Default: `N/A`
  Defines the SQL query to be used to perform authentication and authorization of endpoint registrations. The query is parameterized - that means parameter replacement is made before each query is executed. The following parameters are defined:

  - `%g` - the gatekeeper identifier
  - `%{gkip}` - a gatekeeper IP the request has been received on
  - `%u` - username associated with an endpoint (usually a H.323 ID)
  - `%{callerip}` - caller's IP (the request has been received from - NAT IP for NATted endpoints)
  - `%{aliases}` - a comma separated list of endpoint aliases
  - `%{additive-rrq}` - whether an additive rrq 0 - false 1 - true

  If the query returns no rows, the result is undefined, which basically means failure for `required` rules and "try next" for optional rules. Otherwise, the first result row is examined to determine the result of the authentication request and to get additional information:

  1. The first column is converted into a boolean value (1, T, TRUE, allow, y, yes means true) and is an authentication result (accept/reject).
  2. If the registration is successfully authenticated the remaining columns are examined:
     (a) If there exists a column called `'aliases'`, replace original endpoint aliases with these new ones

(b) If there exists a column called `'billingmode'`, set a billing mode associated with the endpoint (0 - credit,

(c) 0 - debit)

(d) If there exists a column called `'creditamount'`, set account balance associated with the endpoint (this is an arbitrary string)

Query string examples:

```
SELECT 1, 0 AS billingmode, '12.00 USD' AS creditamount
SELECT NOT disabled, assignaliases AS aliases, balance FROM users WHERE h323id = '%u'
SELECT * FROM get_registration_auth('%g', '%u', '%{callerip}', '%{aliases}') AS result(accept, aliases
```

- `NbQuery=SELECT ...`
  Default: `N/A`
  Defines the SQL query to be used to perform authentication and authorization of location requests sent from neighbors. The query is parameterized - that means parameter replacement is made before each query is executed. The following parameters are defined:

  – `%g` - the gatekeeper identifier
  – `%{gkip}` - a gatekeeper IP the request has been received on
  – `%{nbid` - neighbor identifier from the config
  – `%{nbip}` - neighbor IP (the request has been received from)
  – `%{Calling-Station-Id}` - caller's number, if available
  – `%{src-info}` - content of sourceInfo LRQ field, if available
  – `%{Called-Station-Id}` - destination number
  – `%{dest-info}` - content of destinationInfo LRQ field
  – `%{bandwidth}` - requested bandwidth, if present in the LRQ

  If the query returns no rows, the result is undefined, which basically means failure for `required` rules and "try next" for optional rules. Otherwise, the first result row is examined to determine the result of the authentication and to get additional information:

  1. The first column is converted into a boolean value (1, T, TRUE, allow, y, yes means true) and is an authentication result (accept/reject).
  2. If the request is authenticated successfully, remaining columns are examined:
     (a) If there exists a column called `'destination'`, populate the original destinationInfo field with these new aliases - this may affect routing decision, which is made after auth step.

  Query string examples:

```
SELECT active FROM neighbors WHERE name = '%{nbid}' AND ip = '%{nbip}' UNION SELECT 0
```

- `CallQuery=SELECT ...`
  Default: `N/A`
  Define a SQL query to be used to perform authentication and authorization of calls (ARQ and Setup). The query is parameterized - that means parameter replacement is made before each query is executed. The following parameters are defined:

  – `%g` - the gatekeeper identifier
  – `%{gkip}` - a gatekeeper IP the request has been received on
  – `%u` - a username associated with the caller

- %{callerip} - caller's IP (the request has been received from - NAT IP for NATted endpoints)
- %{Calling-Station-Id} - caller's number, if available
- %{Called-Station-Id} - destination number
- %{Dialed-Number} - original destination number (before rewrite)
- %{CallId} - H.323 call identifier (16 hex 8-bit digits)
- %{SrcInfo} - srcInfo field of the ARQ or sourceAddress from Setup
- %{bandwidth} - requested bandwidth, if present in the ARQ
- %{answer} - 1, if the request is an answering ARQ
- %{arq} - 1 for ARQ triggered query, 0 for Setup triggered query

If the query returns no rows, the result is undefined, which basically means failure for `required` rules and "try next" for optional rules. Otherwise, the first result row is examined to determine the authentication result and to get additional information:

1. The first column is converted into a boolean value (1, T, TRUE, allow, y, yes means true) and is an authentication result (accept/reject the call).
2. If the request is authenticated successfully, remaining columns are examined:
   (a) If there exists a column called `'billingmode'`, set a billing mode associated with the endpoint (0 - credit,
   (b) 0 - debit)
   (c) If there exists a column called `'creditamount'`, set account balance associated with the endpoint (this is an arbitrary string)
   (d) If there exists a column called `'credittime'`, use its integer value to set call duration limit
   (e) If there exists a column called `'redirectnumber'`, replace the original destination number with this one. You can put multiple numbers (that correspond to multiple `'redirectip'` entries) separated by a semicolon. You can also specify an outbound number (to be sent to a terminating gateway) by appending it with an '=' to the rewritten number (like 485811001001=1234485811001001)
   (f) If there exists a column called `'redirectip'`, force the call to be sent to the specified IP (one can put multiple destinations separated by a semicolon, that will be used for failover, if failover is activated)
   (g) If there exists a column called `'proxy'`, force the gatekeeper to enable/disable (depends on the 'proxy' column value) RTP proxy for this call
   (h) If there exists a column called `'clientauthid'`, the gatekeeper will store this ID in its call record and send it back on all accounting events. This must be an unsigned integer with a maximum of 64 bits (eg. 'bigint unsigned' in MySQL).
3. If the request is denied, the remaining columns are examined:
   (a) If there exists a column called `'q931cause'`, set a Q.931 cause in a Release Complete to this value
   (b) If there exists a column called `'clientauthid'`, the gatekeeper will store this ID in its call record and send it back on all accounting events. This must be an unsigned integer with a maximum of 64 bits (eg. 'bigint unsigned' in MySQL).

Query string examples:

```
SELECT 1, 360 AS credittime, 0 AS proxy
SELECT * FROM auth_call('%g', '%u', '%{Calling-Station-Id}', '%{callerip}', '%{Called-Station-Id}') AS
SELECT 1, '1234' AS redirectnumber, '192.168.1.1' AS redirectip
```

8.9 Section [PrefixAuth]

The section defines the authentication rule for the `PrefixAuth` module. You can authorized ARQs, LRQs and Setups with this module.

First, the most specific prefix is selected according to the **destinationInfo** field of the received request. Then the request is accepted or rejected according to the matched rules with the most specific netmask. If no matched prefix is found, and the `default` option is specified, the request is accepted or rejected according to that. Otherwise it is rejected or passed to the next authentication module according to the module requirement.

Format:

```
prefix=authrule[|authrule|...]
```

Syntax:

```
<authrule> :=  <result> <authrule>

  <result>    := deny | allow
  <authrule>  := [!]ip:<iprule> | [!]ipv4:<iprule> | [!]ipv6:<iprule> | [!]alias:<aliasrule>
```

Where <`iprule`> can be specified in decimal dot notation or CIDR notation or IPv6 notation, <`aliasrule`> is expressed in regular expression. If the '`!`' flag precedes the rule, the sense is inverted. Rules for IP numbers with `ip:`, `ipv4:`, `ipv6:` all behave the same. The different prefixes are just for documentation and compatibility with older versions of GnuGk.

Example:

```
555=deny ipv4:10.0.0.0/27|allow ipv4:0/0
5555=allow ipv4:192.168.1.1|deny ipv4:192.168.1.0/255.255.255.0
86=deny !ipv4:172.16.0.0/24
09=deny alias:^188884.*
99=deny ipv6:2021:4ad0:ff00:99a::/64
ALL=allow ipv4:0/0|allow ipv6:::/0
```

In this configuration, all endpoints except those from network `10.0.0.0/27` are allowed to call prefix 555 (except 5555). Endpoints from `192.168.1.0/24` are not allowed to call prefix 5555, except `192.168.1.1`. Endpoints **not** from `172.16.0.0/24` are denied to call prefix 86. Endpoints having an alias beginning with 188884 are not allowed to call prefix 09. IPv6 endpoints from 2021:4ad0:ff00:99a::/64 are not allowed to call prefix 99. All other situations are allowed.

8.10 Section [RadAuth]

This section defines configuration settings that enable RADIUS authentication based on H.235 CATs (Cisco Access Tokens) present in RRQ, ARQ RAS requests and Q.931 Setup messages.

- `Servers=SERVER1[:AUTH_PORT[:ACCT_PORT[:SECRET]]];SERVER2[:AUTH_PORT[:ACCT_PORT[:SECRET]]];...`
  Default: `N/A`
  RADIUS servers to be used for authentication. The list can contain an arbitrary number of servers. The order of servers is important, because servers will be queried by the RADIUS module in the given order. If no port information is provided, port number from `DefaultAuthPort` will be used. If no secret is set, the default shared secret from `SharedSecret` is taken. Servers names can be IP addresses or DNS names. IPv6 addresses must always be written in brackets.

Sample `Servers` lines:

```
Servers=192.168.1.1
Servers=192.168.1.1:1645
Servers=192.168.1.1:1645:1646:secret1
Servers=radius1.mycompany.com:1812
Servers=radius1.mycompany.com;radius2.mycompany.com
Servers=radius1.mycompany.com:1812:1813:secret1;radius2.mycompany.com:1812:1813:secret2
Servers=[2501:4f3:61:2143::2]
Servers=[2501:4f3:61:2143::2]:1645
Servers=[2501:4f3:61:2143::2]:1645:1646
Servers=[2501:4f3:61:2143::2]:1645:1646:secret1
Servers=[2501:4f3:61:2143::2]:1645:1646:secret1;[2501:4f3:61:2143::3]:1645:1646:secret2
```

- `LocalInterface=IP_OR_FQDN`
  Default: `N/A`
  The specific local network interface that GnuGk should use in order to communicate with RADIUS servers. This parameter can be useful on NAT machines to restrict which network interfaces are used for RADIUS communication. By default this value is empty and allows RADIUS requests to be sent on any (best suitable) network interface. If you are not sure what you are doing, it is better to leave this option unset.

- `RadiusPortRange=10000-11000`
  Default: `N/A`
  By default, GnuGk allocates ports dynamically as specified by the operating system. If you want to restrict which ports it should use then configure this parameter.

- `DefaultAuthPort=PORT_NO`
  Default: `1812`
  Default port number to be used for RADIUS authentication requests (Access-Request packets). Can be overridden by `Servers` attribute.

- `SharedSecret=SECRET`
  Default: `N/A (empty string)`
  Secret used to authenticate this GnuGk (NAS client) to RADIUS server. It should be a cryptographically strong password. This is the default value used if no server-specific secret is set in the `Servers` configuration option. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in encrypted form and should be created using the `addpasswd` utility.

- `RequestTimeout=TIMEOUT_MS`
  Default: `2000` (milliseconds)
  Timeout (milliseconds) for RADIUS server response to a request sent by GnuGk. If no response is received within this time period, the next RADIUS server is queried.

- `IdCacheTimeout=TIMEOUT_MS`
  Default: `9000` (milliseconds)
  Timeout (milliseconds) for RADIUS request 8-bit identifiers to be unique. If the entire 8-bit identifier range is exhausted within this period, a new client socket (UDP socket) will be allocated by the RADIUS module. Let's take the example: we have approximately 60 RRQs/sec - after ca. 4 seconds 8-bit identifiers range gets exhausted - new socket allocated - after next 4 seconds the second 8-bit identifiers range gets exhausted - third socket allocated - after 9th second identifiers from the pool 1 are available again.

In general, if you have too long a timeout then too many resources will be consumed. If you have too short a timeout, then the RADIUS server may take incoming packets as duplicates and therefore drop them.

- `SocketDeleteTimeout=TIMEOUT_MS`
  Default: `60000` (milliseconds) - 60 s
  Timeout for unused RADIUS sockets to be closed. It is used in conjunction with `IdCacheTimeout` - additional sockets created during heavy gatekeeper load periods for serving incoming requests are closed during idle periods.

- `RequestRetransmissions=NUMBER`
  Default: `2`
  How many times a single RADIUS request is transmitted to every configured RADIUS server (if no response is received). 1 means one transmission attempt and no re-transmission, 2 - single re-transmission, ... Exact retransmission method is defined by `RoundRobinServers` attribute.

- `RoundRobinServers=BOOLEAN`
  Default: `1`
  RADIUS requests retransmission method.

  If set to 1, RADIUS request is transmitted in the following way (until response is received):

  ```
  Server #1 Attempt #1, Server #2 Attempt #1, ..., Server #N Attempt #1
  ...
  Server #1 Attempt #RequestRetransmissions, ..., Server #1 Attempt #RequestRetransmissions
  ```

  If set to 0, the following sequence is preserved:

  ```
  Server #1 Attempt #1, ..., Server #1 Attempt #RequestRetransmissions
  ...
  Server #N Attempt #1, ..., Server #N Attempt #RequestRetransmissions
  ```

- `AppendCiscoAttributes=BOOLEAN`
  Default: `0`
  If set, Cisco Vendor Specific RADIUS attributes are included in RADIUS requests (h323-conf-id,h323-call-origin,h323-call-type).

- `IncludeTerminalAliases=BOOLEAN`
  Default: `1`
  If set, Cisco VSA 'h323-ivr-out' attribute is sent with a list of aliases the endpoint is registering (RRQ.m_terminalAlias). This attribute is provided in order to provide fine control over the list of aliases the endpoint is allowed to register with. Format of this attribute is:

  ```
          Cisco-AV-Pair = "h323-ivr-out=terminal-alias:" alias [,alias] [;]
  Example:
          Cisco-AV-Pair = "h323-ivr-out=terminal-alias:helpdesk,support,77771;"
  ```

- `UseDialedNumber=BOOLEAN`
  Default: `0`
  Select Called-Station-Id number type between the original one (as dialed by the user) - `UseDialedNumber=1` - and the rewritten one - `UseDialedNumber=0`.

8.10.1 [RadAuth] Access-Request Radius Attributes

For RRQs, the following RADIUS attributes are included within Access-Request packets:

- `User-Name`
  H225_RegistrationRequest.tokens[CAT].m_generalID
- `CHAP-Password`
  H225_RegistrationRequest.tokens[CAT].m_random + H225_RegistrationRequest.tokens[CAT].m_challenge
- `CHAP-Challenge`
  H225_RegistrationRequest.tokens[CAT].m_timeStamp
- `NAS-IP-Address`
  GnuGk Home or a particular local network interface set by 'LocalInterface' config parameter
- `NAS-Identifier`
  GnuGk Name
- `NAS-Port-Type`
  Virtual (GnuGk does not have concept of physical ports)
- `Framed-IP-Address`
  An IP address of registering endpoint signaling channel
- `Service-Type`
  Login-User
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, h323-ivr-out`
  A list of aliases an endpoint is registering with (only if IncludeTerminalAliases config option is set)

  NOTE: The list of aliases inside h323-ivr-out is in the following form:
  `h323-ivr-out="h323-ivr-out=terminal-alias:alias1,alias2,...,aliasN;"`
  The h323-ivr-out attribute can be (in future) instantiated multiple times inside a single Access-Request and may also contain variables other than "terminal-alias", so a RADIUS server should be flexible enough with processing of this attribute.

For ARQ and Setup messages, the following RADIUS attributes are included inside Access-Request packets:

- `User-Name`
  ARQ.tokens[CAT].m_generalID
- `CHAP-Password`
  ARQ.tokens[CAT].m_random + ARQ.tokens[CAT].m_challenge
- `CHAP-Challenge`
  ARQ.tokens[CAT].m_timeStamp
- `NAS-IP-Address`
  GnuGk Home or a particular local network interface set by 'LocalInterface' config parameter
- `NAS-Identifier`
  GnuGk Name
- `NAS-Port-Type`
  Virtual (GnuGk does not have concept of physical ports)
- `Framed-IP-Address`
  An IP address of registering endpoint signaling channel
- `Service-Type`
  Login-User (for ARQs from originating endpoint) or Call-Check (for ARQs from answering endpoint)

- `Calling-Station-Id`
  Calling party's number (if available)
- `Called-Station-Id`
  Called party's number
- `(optional) VSA: VendorId=Cisco, h323-conf-id`
  H.323 conference ID from ARQ
- `(optional) VSA: VendorId=Cisco, h323-call-type`
  Call type (fixed value: "h323-call-type=VoIP")
- `(optional) VSA: VendorId=Cisco, h323-call-origin`
  Call origin ("answer","originate")
- `(optional) VSA: VendorId=Cisco, h323-gw-id`
  The same as NAS-Identifier

8.10.2 [RadAuth] Access-Accept Radius Attributes

For RRQs, the following RADIUS attributes are recognized inside Access-Accept packets:

- `VSA: VendorId=Cisco, h323-return-code`
  If present and not 0, the request is rejected. This check is provided to allow interoperability with some poor billing systems, which send Access-Accept with non-zero h323-return-code to reject the call instead of Access-Reject. The attribute can be in the form h323-return-code="1" or h323-return-code="h323-return-code=1". Note that the return code is a string, not an integer.
- `VSA: VendorId=Cisco, h323-billing-model`
  Billing mode for this account. Can be 0 (credit), 1 or 2 (debit). If an endpoint can understand H.225.0 CallCreditServiceControl messages, this information is used to build the message.
- `VSA: VendorId=Cisco, h323-credit-amount`
  A string representing current user's account balance. If an endpoint can understand H.225.0 CallCreditServiceControl messages, this information is used to build the message.
- `VSA: VendorId=Cisco, Cisco-AVPair, h323-ivr-in`
  If present, it is scanned for 'terminal-alias' variable that can contain a list of aliases that should be assigned to the endpoint being registered. All RRQ aliases that do not match this list are removed. The 'disable-codec' variable is also supported to disallow certain codecs for this call. The 'proxy' variable that can contain 'yes' or 'no' for enabling/disabling proxy mode for this call. The format of these attributes is as follows:

  ```
  Cisco-AVPair = "h323-ivr-in=variable:value;[variable:value;]"
  ```

  where the "variable" can be "terminal-alias":

  Cisco-AVPair = "h323-ivr-in=terminal-alias:alias1[,alias2,...];"

  Example 1:

  ```
  RRQ {
          m_terminalAlias = { "myalias", "1234" }
  }

  if RADIUS server returns the following h323-ivr-in:
  ```

```
Access-Accept {
        Cisco-AVPair = "h323-ivr-in=terminal-alias:anotheralias,6789;"
}

the endpoint will get registered with aliases "anotheralias" and "6789".
Also RCF will contain:

RCF {
        m_terminalAlias = { "anotheralias", "6789" }
}
```

Example 2 (add E164 to an existing alias):

```
RRQ {
        m_terminalAlias = { "it_s_me" }
}

if RADIUS server returns the following h323-ivr-in:

Access-Accept {
        Cisco-AVPair = "h323-ivr-in=terminal-alias:it_s_me,48586259732;"
}

RCF will contain:

RCF {
        m_terminalAlias = { "it_s_me", "48586259732" }
}
```

Example 3 (disable G.711 and G.729 codecs):

```
Access-Accept {
        Cisco-AVPair = "h323-ivr-in=codec-disable:g711Ulaw64k;g729;g711Alaw64k;g729AnnexA;"
}
```

Example 4 (enable proxy mode):

```
Access-Accept {
        Cisco-AVPair = "h323-ivr-in=proxy:yes"
}
```

For ARQs, the following RADIUS attributes are recognized within Access-Accept packets:

- `VSA: VendorId=Cisco, h323-return-code`
  If present and not 0, the request is rejected. This check is provided to allow interoperability with some poor billing systems, that send Access-Accept with non-zero h323-return-code to reject the call instead of Access-Reject. The attribute can be in form h323-return-code="1" or h323-return-code="h323-return-code=1". Note that the return code is a string, not an integer.

- `VSA: VendorId=Cisco, h323-billing-model`
  Billing mode for this account. Can be 0 (credit), 1 or 2 (debit). If an endpoint can understand H.225.0 CallCreditServiceControl messages, this information is used to build the message.

- `VSA: VendorId=Cisco, h323-credit-amount`
  A string representing current user account balance. If an endpoint can understand H.225.0 CallCreditServiceControl messages, this information is used to build the message.

- `VSA: VendorId=Cisco, h323-credit-time`
  If present, it enforces maximum call duration (in seconds). The attribute can be in form of h323-credit-time="120" or h323-credit-time="h323-credit-time=120". Note that the return code is a string, not an integer.

- `Session-Timeout`
  If present, it enforces maximum call duration (in seconds). This is a standard RADIUS attribute of integer type.

- `VSA: VendorId=Cisco, h323-redirect-ip-address`
  If present, a call is sent to the IP address present in this attribute. You can put multiple destinations separated with a semicolon.

- `VSA: VendorId=Cisco, h323-redirect-number`
  If present, a called station id is rewritten to this number. You can put multiple numbers separated by a semicolon. For each number you can also specify an outbound number (that is sent to a terminating gateway) by appending it with a '='.

NOTE: If both Session-Timeout and h323-credit-time are present, the smaller value is used.

NOTE: If multiple failover mechanisms are specified, eg. multiple numbers in h323-redirect-number **and** multiple IPs in h323-redirect-ip-address, there is no guarantee that the the first number is used for the first IP and the 2nd number for the 2nd IP. This will usually the case, but for example when a capacity limit disables one IP, the association will change.

8.11 Section [RadAliasAuth]

This section defines configuration settings that enable RADIUS authentication based on endpoint aliases and/or IP addresses present in a RRQ RAS, ARQ RAS or Q.931 Setup request. This authentication scheme is useful both for endpoints registered at the gatekeeper (ARQ, RRQ) and calls from unregistered endpoints (Setup).

- `Servers=SERVER1[:AUTH_PORT[:ACCT_PORT[:SECRET]]];SERVER2[:AUTH_PORT[:ACCT_PORT[:SECRET]]];...`
  Default: `N/A`
  RADIUS servers to be used for RAS requests authentication. This list can contain an arbitrary number of servers. The order of servers is important, because servers will be queried by the RADIUS module in the given order. If no port information is specified, the port number from `DefaultAuthPort` will be used. If no secret is set, the default shared secret from `SharedSecret` is used. Servers can be IP addresses or DNS names.

  Example:

  ```
  Servers=192.168.3.1:1645;192.168.3.2:1812:1813:mysecret;radius.mycompany.com
  ```

- `LocalInterface=IP_OR_FQDN`
  Default: `N/A`
  Specific local network interface that GnuGk should use in order to communicate with RADIUS servers. This parameter can be useful on NAT machines to restrict number of network interfaces used for RADIUS communication. By default this value is empty and allows RADIUS requests to be sent on any (best suitable) network interface. If you are not sure what you are doing, it is better to leave this option unset.

- `RadiusPortRange=10000-11000`
  Default: `N/A`

By default (if this option is not set) RADIUS client allocates ports dynamically as specified by the operating system. If you want to restrict RADIUS client to use ports from a particular range only - set this parameter.

- `DefaultAuthPort=PORT_NO`
  Default: `1812`
  Default port number to be used for RADIUS authentication requests (Access-Request packets), if not overridden by `Servers` attribute.

- `SharedSecret=SECRET`
  Default: `N/A (empty string)`
  Secret used to authenticate this GnuGk (NAS client) to RADIUS server. It should be a cryptographically strong password. This is the default value used, if no server-specific secret is set in the `Servers`. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in encrypted form and should be created using the `addpasswd` utility.

- `RequestTimeout=TIMEOUT_MS`
  Default: `2000` (milliseconds)
  Timeout (milliseconds) for RADIUS server response to a request sent by GnuGk. If no response is received within this time period, next RADIUS server is queried.

- `IdCacheTimeout=TIMEOUT_MS`
  Default: `9000` (milliseconds)
  Timeout (milliseconds) for RADIUS request 8-bit identifiers to be unique. If all 8-bit identifier range is exhausted within this period, new client socket (UDP socket) is allocation by RADIUS module. Let's take the example: we have approximately 60 RRQs/sec - after ca. 4 seconds 8-bit identifiers range gets exhausted - new socket allocated - after next 4 seconds the second 8-bit identifiers range gets exhausted - third socket allocated - after 9th second identifiers from the pool 1 are available again - ... . In general, too long timeout - too much resources consumed, too short timeout - RADIUS server may take incoming packets as duplicated and therefore drop it.

- `SocketDeleteTimeout=TIMEOUT_MS`
  Default: `60000` (milliseconds) - 60 s
  Timeout for unused RADIUS sockets to be closed. It is used in conjunction with `IdCacheTimeout` - additional sockets created during heavy gatekeeper load periods for serving incoming requests are closed during idle periods.

- `RequestRetransmissions=NUMBER`
  Default: `2`
  How many times a single RADIUS request is transmitted to every configured RADIUS server (if no response is received). 1 means no retransmission, 2 - single retransmission, ... Exact retransmission method is defined by `RoundRobinServers` attribute.

- `RoundRobinServers=BOOLEAN`
  Default: `1`
  RADIUS requests retransmission method.

  If set to 1, RADIUS request is transmitted in the following way (until response is received):

  ```
  Server #1 Attempt #1, Server #2 Attempt #1, ..., Server #N Attempt #1
  ...
  Server #1 Attempt #RequestRetransmissions, ..., Server #1 Attempt #RequestRetransmissions
  ```

  If set to 0, the following sequence is preserved:

```
Server #1 Attempt #1, ..., Server #1 Attempt #RequestRetransmissions
...
Server #N Attempt #1, ..., Server #N Attempt #RequestRetransmissions
```

- `AppendCiscoAttributes=BOOLEAN`
  Default: `1`
  If set, Cisco Vendor Specific RADIUS attributes are included in RADIUS requests (h323-conf-id,h323-call-origin,h323-call-type).

- `IncludeTerminalAliases=BOOLEAN`
  Default: `1`
  If set, Cisco VSA 'h323-ivr-out' attribute is sent with a list of aliases the endpoint is registering (RRQ.m_terminalAlias). This attribute is provided in order to provide fine control over the list of aliases the endpoint is allowed to register with. Format of this attribute is:

```
        Cisco-AV-Pair = "h323-ivr-out=terminal-alias:" alias [,alias] [;]
Example:
        Cisco-AV-Pair = "h323-ivr-out=terminal-alias:helpdesk,support,77771;"
```

- `EmptyUsername`
  Default: `N/A`
  If this parameter is set, the value is used if the call doesn't provide a username for authentication.

- `FixedUsername`
  Default: `N/A`
  If this parameter is set, it overwrites a value of User-Name RADIUS attribute for outgoing RADIUS request. That means every Access-Request will be authenticated as for user `FixedUsername`.

- `FixedPassword`
  Default: `N/A`
  If not set, User-Password is a copy of User-Name. For example, if User-Name is 'john' then User-Password will also be set to 'john'. Setting this parameter overrides this behavior and User-Password attribute will be always set to the value of `FixedPassword`. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in encrypted form and should be created using the `addpasswd` utility.

  Example 1:

  ```
  (Neither FixedUsername nor FixedPassword set)
  ```

  All endpoints will be authenticated using their alias as the username and the password. That means, for example, endpoint 'EP1' will be authenticated with the username 'EP1 and the password 'EP1'.

  Example 2:

  ```
  (FixedUsername not set)
  FixedPassword=ppp
  ```

  All endpoints will be authenticated using their alias and the password 'ppp'.

  Example 3:

  ```
  FixedUsername=ppp
  FixedPassword=ppp
  ```

  All endpoints will be authenticated using the username 'ppp' and the password 'ppp'.

- `UseDialedNumber=BOOLEAN`
  Default: `0`
  Select Called-Station-Id number type between the original one (as dialed by the user) - `UseDialedNumber=1` - and the rewritten one - `UseDialedNumber=0`.

8.11.1 [RadAliasAuth] Access-Request Radius Attributes

For RRQs, the same attributes as with RadAuth are sent, with an exception of username/password attributes (CHAP-Password, CHAP-Challenge, User-Name):

- `User-Name`
  Either an endpoint alias from RRQ or a value of FixedUsername config parameter. If no alias is present, an IP address is used
- `User-Password`
  Either the same as User-Name or a value of FixedPassword config parameter

For ARQ and Setup messages, the same attributes as with RadAuth are sent, with an exception of username/password attributes (CHAP-Password, CHAP-Challenge, User-Name):

- `User-Name`
  Either an endpoint alias or a value of FixedUsername config parameter
- `User-Password`
  Either the same as User-Name or a value of FixedPassword config parameter

8.11.2 [RadAliasAuth] Access-Accept Radius Attributes

Exactly the same attributes are recognized as with RadAuth module.

8.12 Section [CapacityControl]

This section contains a set of rules for controlling inbound call volume depending on various conditions. In order for this module to work, CapacityControl authentication and accounting modules have to be enabled like this:

```
[Gatekeeper::Auth]
CapacityControl=required;Setup

[Gatekeeper::Acct]
CapacityControl=required;start,stop
```

A capacity rule can be matched by a caller's IP, caller's H.323 ID and/or caller's number (CLI) - in the order specified. In addition, the match can be narrowed by specifying a called number pattern. This module works by keeping lists of current call volume for each inbound route (rule) - this is done by having `CapacityControl` accounting module configured to add/remove active calls from matching routes. The `CapacityControl` authentication module checks rules and accepts/rejects a call based on current/max call volume for a matching inbound route.

Format for an inbound route rule:

`[ip:CALLER_IP|h323id:CALLER_H323ID|cli:CALLER_NUMBER]=[CALLED NUMBER REGEX PATTERN] MAX_CAPACITY`

`ip:`, `h323id:` and `cli:` prefixes define rule type. An inbound call will be matched either by caller's IP, H.323ID or CLI. The optional `CALLED NUMBER REGEX PATTERN` is a regular expression that the called number should match to apply this rule to. `MAX_CAPACITY` is maximum number of active calls for this route.

The rules are match in the following order:

- IP rules
- H.323ID rules
- CLI rules

The longest match in the first matching category is used.

Example 1:

```
[CapacityControl]
ip:192.168.1.0/24=30
ip:any=120
```

These rules tell that the 192.168.1.0/24 subnet can send up to 30 concurrent calls, while all other IPs can send up to 120 concurrent calls.

Example 2:

```
[CapacityControl]
%r1% cli:1001=30
%r2% cli:1001=^48(50|51) 5
```

These rules limit caller with CLI 1001 to send up to 5 calls to 4850/4851 destinations and up to 30 calls to other destinations. %r1% and %r2% are special constructs to allow having the same `cli:1001` config key more than once.

8.13 Section [GkH350::Settings]

This section defines the LDAP server and standard H.350 directory operating parameters to be used.

- `ServerName=127.0.0.1`
  Default: `127.0.0.1` The LDAP server IP address.
- `ServerPort=389`
  Default: `389` The LDAP server's TCP port (usually 389).
- `StartTLS=1`
  Default: `0` Use StartTLS to encrypt the LDAP connection. (This option requires GnuGk to be compiled with PTLib 2.11.0 or higher.)
- `SearchBaseDN=ou=commObjects,dc=gnugk,dc=org`
  Default: `N/A` Entry point into the server's H.350 directory structure. Searches are only made below this root node.
- `BindUserDN=cn=admin,dc=gnugk,dc=org`
  Default: `N/A` The distinguished name the gatekeeper uses to bind to the LDAP server. Leave empty if you want to access the LDAP server anonymously.
- `BindUserPW=secret`
  Default: `N/A` If you specified `BindUserDN`, then specify the corresponding password to be used for binding here. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in an encrypted form and should be created using the `addpasswd` utility.
- `BindAuthMode=simple`
  Default: `simple` Bind Authentication method choices are `simple,sasl,kerberos`

- `ServiceControl=1`
  Default: `0`
  Use RRQ/RCF service control field to advise an endpoint of the H.350 directory and searchDN to use for white page lookups.

- `AssignedAliases=1`
  Default: `0`
  Use H.350.1 to advise endpoints of their assigned aliases.

- `GatekeeperDiscovery=1`
  Default: `0`
  Use H.350.1 to resolve on GRQ/GCF the registering endpoints assigned gatekeeper (h323IdentityGKDomain).

8.14 Section [GkLDAP::Settings]

This section defines the LDAP server and connection parameters for the LDAP authentication modules (LDAPAliasAuth and LDAPPasswordAuth) and the LDAP routing policy ('ldap').

- `ServerName=127.0.0.1`
  Default: `127.0.0.1` The LDAP server IP address.

- `ServerPort=389`
  Default: `389` The LDAP server's TCP port (usually 389).

- `StartTLS=1`
  Default: `0` Use StartTLS to encrypt the LDAP connection. (This option requires GnuGk to be compiled with PTLib 2.11.0 or higher.)

- `SearchBaseDN=ou=commObjects,dc=gnugk,dc=org`
  Default: `N/A` Entry point into the server's H.350 directory structure. Searches are only made below this root node.

- `BindUserDN=cn=admin,dc=gnugk,dc=org`
  Default: `N/A` The distinguished name the gatekeeper uses to bind to the LDAP server. Leave empty if you want to access the LDAP server anonymously.

- `BindUserPW=secret`
  Default: `N/A` If you specified `BindUserDN`, then specify the corresponding password to be used for binding here. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in an encrypted form and should be created using the `addpasswd` utility.

- `BindAuthMode=simple`
  Default: `simple` Bind Authentication method choices are `simple,sasl,kerberos`

Example:

```
[GkLDAP::Settings]
ServerName=192.168.1.1
BindAuthMode=simple
SearchBaseDN=dc=gnugk,dc=org
BindUserDN=cn=admin,dc=gnugk,dc=org
BindUserPW=secret
```

8.15 Section [GkLDAP::LDAPAttributes]

With this section you can map the LDAP attributes GnuGk queries to your LDAP schema.

- `CallDestination=H323IP`
  Default: `voIPIpAddress` The IP where calls for the LDAP entity should be routed.
- `H323ID=mail`
  Default: `mail` The H.323 alias for the LDAP entity. This attribute is used to find the called entity in the LDAP schema.
- `H235PassWord=secret`
  Default: `none` The password attribute to use in LDAPPasswordAuth.
- `IPAddress=voIPIpAddress`
  Default: `voIPIpAddress` The H.323 IP for the LDAP entity. This attribute is used to find the called entity in the LDAP schema.
- `TelephoneNo=telephoneNumber`
  Default: `telephoneNumber` The phone number for the LDAP entity. This attribute is used to find the called entity in the LDAP schema.

Example:

```
[GkLDAP::LDAPAttributeNames]
IPAddress=voIPIpAddress
H235PassWord=plaintextPassword
H323ID=sn
TelephonNo=telephoneNumber
CallDestination=roomNumber
```

8.16 Section [LuaAuth]

This section configures the scripts for LUA authentication.

For details on the LUA language, please see *http://www.lua.org/docs.html* `<http://www.lua.org/docs.html>`.

- `RegistrationScript=`*`script`*
  Default: `N/A` LUA script to be used for RRQs. Short scripts can be put directly into the config file. For longer scripts use RegistrationScriptFile=.
- `RegistrationScriptFile=/path/to/script`
  Default: `N/A` LUA script to be used for RRQs.
- `CallScript=`*`script`*
  Default: `N/A` LUA script to be used for ARQs and Setups. Short scripts can be put directly into the config file. For longer scripts use CallScriptFile=.
- `CallScriptFile=/path/to/script`
  Default: `N/A` LUA script to be used for ARQs and Setups.

Registration authentication scripts can use the following variables

- `username` - a username associated with the caller

- `callerIP` - caller's IP (the request has been received from - NAT IP for NATted endpoints)
- `aliases` - the list of aliases in the registration
- `messageType` - always "RRQ" for registrations

Call authentication scripts can use the following variables

- `source` -
- `calledAlias` -
- `calledIP` -
- `caller` -
- `callingStationId` - caller's number, if available
- `callid` - the call ID
- `messageType` - either "ARQ" or "Setup"

The scripts have to store the authentication result in the variable *result*. Possible values are "OK", "FAIL" and "NEXT"; everything else will be treated as "FAIL".

Example:

```
[LuaAuth]
; only let use "boss" register
RegistrationScript=if (username == "boss") then result = "OK" else result = "FAIL" end
; call authentication is in a script file
CallScriptFile=/path/to/callauth.lua
```

8.17 Section [GeoIPAuth]

This section configure the GeoIP authentication.

- `Database=/path/to/GeoIP.dat`
  Default: `N/A` Path to the Maxmind GeoIP database. You can download the latest copy of the GeoIP database from *http://dev.maxmind.com/geoip/legacy/geolite/*
- `AllowedCountries=US,CA,PRIVATE`
  Default: `N/A` Define which countries you want to accept messages from. The special country 'PRIVATE' denotes ptivate IPs.

9 Accounting Configuration

The following sections in the config file can be used to configure accounting.

9.1 Section [Gatekeeper::Acct]

This section defines a list of modules which may be used to perform accounting. The accounting function can be used for logging gatekeeper on/off events and call start/stop/update events. Each accounting module logs received events to module-specific storage. The various storage options include plain text file, RADIUS server and many more. The configuration is very similar to the one for gatekeeper authentication (see 8.1 ([Gatekeeper::Auth])).

All CDRs are also sent to the status port and can be used by external applications.

Syntax:

```
acctmod=actions

 <acctmod> := FileAcct | RadAcct | SQLAcct | StatusAcct | SyslogAcct | CapacityControl | ...
 <actions> := <control>[;<event>,<event>,...]
 <control> := optional | required | sufficient | alternative
 <event>   := start | stop | alert | connect | update | register | unregister | on | off
```

The event list tells the gatekeeper which events should trigger logging with the given accounting module (if an event type is supported by the module):

- `start` - a call has been started and a Setup message has been received (only available in routed mode)
- `alert` - a call is alerting (only available in routed mode)
- `connect` - a call has been connected (only available in routed mode)
- `update` - a call is active and a periodic update is performed to reflect the new call duration. The frequency of these updates is determined by the **AcctUpdateInterval** variable from the 12.1 ([CallTable]) section
- `register` - an endpoint has registered
- `unregister` - an endpoint has unregistered
- `stop` - a call has been disconnected (removed from the gatekeeper call table)
- `on` - the gatekeeper has been started
- `off` - the gatekeeper has been shut down

An event logged by a module may results in one of three result codes: **ok**, **fail**, **next**.

- `ok` - the event has been logged successfully by this module
- `fail` - the module failed to log the event
- `next` - the event has not been logged by this module, because the module is not configured for/does not support this event type

Accounting modules can be stacked to log events by multiple modules or to create failover setups. The **control** flag for each module, along with result codes, define what is the final status of the event processing by the entire module stack. If the final result is **failure**, some special actions may take place. Currently, if a call **start** event logging fails, the call is disconnected immediately. The following **control** flags are recognized:

- `required` - if the module fails to log an event, the final status is set to failure and the event is passed down to any remaining modules.
- `optional` - the module tries to log an event, but the final status is not affected by success or failure (except when the module is last on the list). The event is always passed down to any remaining modules.
- `sufficient` - the module determines the final status. If an event is logged successfully, no remaining modules are processed. Otherwise the final status is set to failure and the event is passed down to any remaining modules.
- `alternative` - if the module logs an event successfully, no remaining modules are processed. Otherwise the final status is not modified and the event is passed down to any remaining modules.

Currently supported accounting modules:

- `FileAcct` A plain Call Detail Report ("CDR") text file logger. It outputs CDR status data to a specified text file. This module only supports the **stop** accounting event. Configuration settings are read from 9.3 ([FileAcct]) section.
- `RadAcct` This module performs RADIUS accounting. It supports the event types start, stop, update, on, off. See section 9.4 ([RadAcct]) for configuration details.
- `SQLAcct` This module performs direct SQL accounting. It supports (start, connect, stop, update, alert, register, unregister) event types. See section 9.5 ([SQLAcct]) for configuration details.
- `StatusAcct` This module logs all accounting events on the status port. It can be used to interface to external application in real-time. It supports (start, connect, stop, update, alert, register, unregister) event types. See section 9.6 ([StatusAcct]) for configuration details.
- `SyslogAcct` This module logs all accounting events to the Unix syslog. It supports (start, connect, stop, update) event types. See section 9.7 ([SyslogAcct]) for configuration details.
- `CapacityControl` This module performs inbound call volume logging, required for the `CapacityControl` authentication module to work correctly. See the section 8.12 ([CapacityControl]) for details.
- `default` This is a special pseudo module - it is used to set the final status if other modules have not determined it. The format is as follows:

  Syntax:

  ```
          default=<status>[;<event>,<event>,...]
          <status> := accept | fail
          <event>  := start | stop | alert | connect | update | register | unregister | on | off
  ```

Sample configuration #1 (try to log call start/stop with RADIUS server, and always write a CDR to a text file):

Example:

```
RadAcct=optional;start,stop
FileAcct=required
```

Sample configuration #2 (try to log call start/stop with RADIUS server, if it fails use a CDR log file):

Example:

```
RadAcct=alternative;start,stop
FileAcct=sufficient;stop
default=accept
```

The **default** rule is required here to prevent calls from being rejected because of RadAcct start event logging failure. If RadAcct responds with a **fail** return code, it is passed down to the FileAcct module. The FileAcct module does not support **start** events, so it returns a **next** code. If there were no **default** rule, the final status would be failure, because no module has been able to log the event.

Sample configuration #3 (always log call start and stop events with RADIUS server, if it fails for call stop event, use a CDR file to store call info):

Example:

```
RadAcct=alternative;start,stop
FileAcct=sufficient;stop
default=fail;start
```

The **default** rule is optional here. If RadAcct returns a **fail** code for the **start** event, the code is passed to the FileAcct module. The FileAcct module does not support **start** events, so it returns **next** return code. The **default** rule ensures that the call is disconnected if the call start event could not be logged with RadAcct. However, we still want to store a CDR in a text file in case the RADIUS server is down when the call disconnects, so we can fetch call duration into a billing system later.

9.2 Customizing CDR strings

Most accounting modules let you customize the CDR data they store. They use a common set of parameters to define the CDR string.

Parameters are specified using `%` character and can be one letter (like `%n`) or longer (like `%{CallId}`). Any remaining characters that are not parameter names are simply copied to the final CDR string. The following parameters are recognized:

- `%g` - gatekeeper name
- `%n` - call number (not unique after gatekeeper restart)
- `%d` - call duration (seconds)
- `%t` - total call duration (from Setup to Release Complete)
- `%c` - Q.931 disconnect cause (decimal integer) as originally received
- `%{cause-translated}` - Q.931 disconnect cause (decimal integer) after translation rules

- `%r` - who disconnected the call (-1 - unknown, 0 - the gatekeeper, 1 - the caller, 2 - the callee)
- `%p` - PDD (Post Dial Delay) in seconds
- `%s` - unique (for this gatekeeper) session identifier (Acct-Session-Id)
- `%u` - H.323 ID of the calling party
- `%{gkip}` - IP address of the gatekeeper
- `%{CallId}` - H.323 call identifier (16 hex 8-bit digits)
- `%{ConfId}` - H.323 conference identifier (16 hex 8-bit digits)
- `%{CallLink}` - Linked H.323 conference identifier (billing account for H.450 call transfer)
- `%{setup-time}` - timestamp string for Q.931 Setup message
- `%{alerting-time}` - timestamp string for Q.931 Alerting message
- `%{connect-time}` - timestamp string for a call connected event
- `%{disconnect-time}` - timestamp string for a call disconnect event
- `%{ring-time}` - time a remote phone was ringing for (from Alerting till Connect or Release Complete)
- `%{caller-ip}` - signaling IP address of the caller
- `%{caller-port}` - signaling port of the caller
- `%{callee-ip}` - signaling IP address of the called party
- `%{callee-port}` - signaling port of the called party
- `%{src-info}` - a colon separated list of source aliases
- `%{dest-info}` - a colon separated list of destination aliases
- `%{Calling-Station-Id}` - calling party number
- `%{Called-Station-Id}` - called party number (rewritten)
- `%{Dialed-Number}` - dialed number (as received from the calling party)
- `%{caller-epid}` - endpoint identifier of the calling party
- `%{callee-epid}` - endpoint identifier of the called party
- `%{call-attempts}` - number of attempts to establish the calls (with failover this can be > 1)
- `%{last-cdr}` - is this the last CDR for this call ? (0 / 1) only when using failover this can be 0
- `%{media-oip}` - caller's RTP media IP (only for H.245 routed/tunneled calls, not for encrypted calls or when firewall traversal is used)
- `%{codec}` - audio codec used during the call (only for H.245 routed/tunneled calls)
- `%{bandwidth}` - bandwidth for this call
- `%{client-auth-id}` - a 64 bit integer ID provided to GnuGk when authenticating the call (through SQLAuth)
- `%{caller-vendor}` - vendor and version info of the calling endpoint
- `%{callee-vendor}` - vendor and version info of the called endpoint
- `%{sinfo-ip}` - IP from Sorenson SInfo (only available when TranslateSorensonSourceInfo=1)

9.3 Section [FileAcct]

This accounting module writes CDR lines to a specified text file. The CDR format can be a standard one (the same as displayed by the status interface) or a customized one (using parametrized query string).

- `DetailFile=FULL_PATH_AND_FILENAME`
  Default: `N/A`
  A full path to the CDR plain text file. If a file with the given name already exists, new CDRs will be appended at the end of the file.

- `StandardCDRFormat=0`
  Default: `1`
  Use a CDR format compatible with the status interface CDR format (`1`) or build a custom CDR string from the **CDRString** parametrized string.

  The StandardCDRFormat is equivalent to this definition:

  ```
  TimestampFormat=RFC822
  CDRString=CDR|%n|%{CallId}|%d|%{connect-time}|%{disconnect-time}|%{caller-ip}:%{caller-port}|%{
  ```

- `CDRString=%s|%g|%u|%{Calling-Station-Id}|%{Called-Station-Id}|%d|%c`
  Default: `N/A`
  If **StandardCDRFormat** is disabled (0) or not specified at all, this parametrized string instructs the gatekeeper on how to build a custom CDR. You can use the 9.2 (common CDR parameters) to define what to include into your CDRs.

- `TimestampFormat=Cisco`
  Default: `N/A`
  Format of timestamp strings printed in CDR strings. If this setting is not specified, the global one from the main gatekeeper section is used.

- `Rotate=hourly | daily | weekly | monthly | L... | S...`
  Default: `N/A`
  If set, the CDR file will be rotated based on this setting. Hourly rotation enables rotation once per hour, daily - once per day, weekly - once per week and monthly - once per month. An exact rotation moment is determined by a combination of RotateDay and RotateTime. During rotation, an existing file is renamed to CURRENT_FILENAME.YYYYMMDD-HHMMSS, where YYYYMMDD-HHMMSS is replaced with the current timestamp, and new CDRs are logged to an empty file.
  In addition, rotation per number of CDRs written (L...) and per file size (S...) is supported. The `L` prefix specifies a number of CDR lines written, the `S` prefix specifies CDR file size. `k` and `m` suffixes can be used to specify thousands (kilobytes) and millions (megabytes).

  Example 1 - no rotation:

  ```
  [FileAcct]
  DetailFile=/var/log/gk/cdr.log
  ```

  Example 2 - rotate every hour (00:45, 01:45, ..., 23:45):

  ```
  [FileAcct]
  DetailFile=/var/log/gk/cdr.log
  Rotate=hourly
  RotateTime=45
  ```

Example 3 - rotate every day at 23:00 (11PM):

```
[FileAcct]
DetailFile=/var/log/gk/cdr.log
Rotate=daily
RotateTime=23:00
```

Example 4 - rotate every Sunday at 00:59:

```
[FileAcct]
DetailFile=/var/log/gk/cdr.log
Rotate=weekly
RotateDay=Sun
RotateTime=00:59
```

Example 5 - rotate on the last day of each month:

```
[FileAcct]
DetailFile=/var/log/gk/cdr.log
Rotate=monthly
RotateDay=31
RotateTime=23:00
```

Example 6 - rotate per every 10000 CDRs:

```
[FileAcct]
DetailFile=/var/log/gk/cdr.log
Rotate=L10000
```

Example 7 - rotate per every 10 kilobytes:

```
[FileAcct]
DetailFile=/var/log/gk/cdr.log
Rotate=S10k
```

9.4 Section [RadAcct]

This accounting module sends accounting data to a RADIUS server. Module configuration is almost the same as for RADIUS authenticators (see 8.10 ([RadAuth]) and 8.11 ([RadAliasAuth]) for more details on the parameters).

- `Servers=SERVER1[:AUTH_PORT:ACCT_PORT[:SECRET]];SERVER2[:AUTH_PORT:ACCT_PORT[:SECRET]];...`
  Default: `N/A`
  RADIUS servers to send accounting data to. If no port information is given, a port number from `DefaultAcctPort` is be used. If no secret is set, the default shared secret from `SharedSecret` is used. Server names may be specified by IP address or DNS name. IPv6 addresses must always be written in brackets.

  Sample `Servers` lines:

  ```
  Servers=192.168.1.1
  Servers=192.168.1.1:1645:1646
  ```

```
Servers=192.168.1.1:1645:1646:secret1
Servers=radius1.mycompany.com:1812:1813
Servers=radius1.mycompany.com;radius2.mycompany.com
Servers=radius1.mycompany.com:1812:1813:secret1;radius2.mycompany.com:1812:1813:secret2
Servers=[2501:4f3:61:2143::2]
Servers=[2501:4f3:61:2143::2]:1645
Servers=[2501:4f3:61:2143::2]:1645:1646
Servers=[2501:4f3:61:2143::2]:1645:1646:secret1
Servers=[2501:4f3:61:2143::2]:1645:1646:secret1;[2501:4f3:61:2143::3]:1645:1646:secret2
```

- `LocalInterface=IP_OR_FQDN`
  Default: `N/A`
  Specific local network interface that *GnuGk* should use in order to communicate with RADIUS servers.

- `RadiusPortRange=10000-11000`
  Default: `N/A`
  By default (if this option is not set) GnuGk allocates ports dynamically as specified by the operating system. In order to restrict the ports which GnuGk will use then configure this parameter appropriately.

- `DefaultAcctPort=PORT_NO`
  Default: `1813`
  Default port number to be used for RADIUS accounting requests. May be overridden by the `Servers` attribute.

- `SharedSecret=SECRET`
  Default: `N/A (empty string)`
  A secret used to authenticate this GnuGk (NAS client) to a RADIUS server. It should be a cryptographically strong password. This is the default value used if no server-specific secret is set in the `Servers`. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in encrypted form and should be created using the `addpasswd` utility.

- `RequestTimeout=TIMEOUT_MS`
  Default: `2000` (milliseconds)
  Timeout (milliseconds) for a RADIUS server response to a request sent by GnuGk. If no response is received within this time period, then the next RADIUS server is queried.

- `IdCacheTimeout=TIMEOUT_MS`
  Default: `9000` (milliseconds)
  Timeout (milliseconds) for RADIUS request 8-bit identifiers to be unique.

- `SocketDeleteTimeout=TIMEOUT_MS`
  Default: `60000` (milliseconds) - 60 s
  Timeout for unused RADIUS sockets to be closed.

- `RequestRetransmissions=NUMBER`
  Default: `2`
  How many times a single RADIUS request is transmitted to every configured RADIUS server (if no response is received).

- `RoundRobinServers=BOOLEAN`
  Default: `1`
  RADIUS requests retransmission method.

  If set to 1, RADIUS request is transmitted in the following way (until response is received):

```
Server #1 Attempt #1, Server #2 Attempt #1, ..., Server #N Attempt #1
...
Server #1 Attempt #RequestRetransmissions, ..., Server #1 Attempt #RequestRetransmissions
```

If set to 0, the following sequence is preserved:

```
Server #1 Attempt #1, ..., Server #1 Attempt #RequestRetransmissions
...
Server #N Attempt #1, ..., Server #N Attempt #RequestRetransmissions
```

- `AppendCiscoAttributes=BOOLEAN`
  Default: `0`
  If set, Cisco Vendor Specific RADIUS attributes are included in RADIUS requests (h323-conf-id,h323-call-origin,h323-call-type).

- `TimestampFormat=ISO8601`
  Default: `N/A`
  Format of timestamp strings sent in RADIUS attributes. If this setting is not specified, the global one from the main gatekeeper section is applied.

- `UseDialedNumber=BOOLEAN`
  Default: `0`
  Select Called-Station-Id number type between the original one (as dialed by the user) - `UseDialedNumber=1` - and the rewritten one - `UseDialedNumber=0`.

9.4.1 [RadAcct] Accounting-Request RADIUS Attributes

For an Accounting-Request, the following RADIUS attributes are included within Accounting-Request packets. Each attribute is followed by a list of accounting event types.

- `Acct-Status-Type (start,update,stop,on,off)`
  The accounting event type (Start, Interim-Update, Stop, Accounting-On, Accounting-Off).

- `NAS-IP-Address (start,update,stop,on,off)`
  An IP address of the gatekeeper.

- `NAS-Identifier (start,update,stop,on,off)`
  The gatekeeper identifier (Name= gk parameter).

- `NAS-Port-Type (start,update,stop,on,off)`
  Fixed value Virtual.

- `Service-Type (start,update,stop)`
  Fixed value Login-User.

- `Acct-Session-Id (start,update,stop)`
  A unique accounting session identifier string.

- `User-Name (start,update,stop)`
  Calling party's account name.

- `Framed-IP-Address (start,update,stop)`
  An IP address for the calling party. Either an endpoint call signaling address or a remote socket address for the signaling channel.

- `Acct-Session-Time (update,stop)`
  Call duration (seconds) - for interim-update events this is the actual duration.

- `Calling-Station-Id (start,update,stop)`
  Calling party's number.
- `Called-Station-Id (start,update,stop)`
  Called party's number.
- `(optional) VSA: VendorId=Cisco, h323-gw-id (start,update,stop)`
  The same as NAS-Identifier.
- `(optional) VSA: VendorId=Cisco, h323-conf-id (start,update,stop)`
  H.323 Conference ID for the call.
- `(optional) VSA: VendorId=Cisco, h323-call-origin (start,update,stop)`
  Fixed string "proxy".
- `(optional) VSA: VendorId=Cisco, h323-call-type (start,update,stop)`
  Fixed string "VoIP".
- `(optional) VSA: VendorId=Cisco, h323-setup-time (start,update,stop)`
  Timestamp when the Q.931 Setup message has been received by the gk.
- `(optional) VSA: VendorId=Cisco, h323-connect-time (update,stop)`
  Timestamp when the call has been connected (Q.931 Setup message has been received or ACF has been sent in direct signaling mode).
- `(optional) VSA: VendorId=Cisco, h323-disconnect-time (stop)`
  Timestamp when the call has been disconnected (ReleaseComplete or DRQ has been received).
- `(optional) VSA: VendorId=Cisco, h323-disconnect-cause (stop)`
  Q.931 two digit hexadecimal disconnect cause.
- `(optional) VSA: VendorId=Cisco, h323-remote-address (start,update,stop)`
  An IP address of the called party (if known).
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, h323-ivr-out (start, update, stop)`
  h323-call-id variable that contains an H.323 Call Identifier. The syntax is as follows: "h323-ivr-out=h323-call-id:123FDE 12348765 9abc1234 12".
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, h323-ivr-out (start, update, stop)`
  rewritten-e164-num contains the rewritten called party's number (independent of the setting of the UseDialedNumber switch).
- `Acct-Delay-Time (start,update,stop)`
  Amount of time (seconds) the gatekeeper is trying to send the request. Currently always 0.
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, h323_rtp_proxy (stop)`
  Proxy mode of call (0=off, 1=on)
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTP_source_IP (stop)`
  RTCP source report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTP_destination_IP (stop)`
  RTCP source report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_source_packet_count (stop)`
  RTCP source report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_source_packet_lost (stop)`
  RTCP source report data

- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_source_jitter (stop)`
  RTCP source report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_source_sdes_XXX (stop)`
  RTCP source report data (for each source description (sdes))
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_destination_packet_count (stop)`
  RTCP destination report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_destination_packet_lost (stop)`
  RTCP destination report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_destination_jitter (stop)`
  RTCP destination report data
- `(optional) VSA: VendorId=Cisco, Cisco-AVPair, RTCP_destination_sdes_XXX (stop)`
  RTCP destination report data (for each source description (sdes))

9.4.2 [RadAcct] Accounting-Response Radius Attributes

The gatekeeper ignores all attributes present in Accounting-Response Radius packets.

9.5 Section [SQLAcct]

This accounting module stores accounting information directly to a SQL database. Many configuration settings are common with other SQL modules.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `StartQuery=INSERT ...`
  Default: `N/A`
  Defines SQL query used to insert a new call record to the database. The query is parametrized - that means parameter replacement is made before each query is executed. You can use the 9.2 (common CDR parameters) to define what to include into your CDRs.

  Sample query string:

  ```
  INSERT INTO call (gkname, sessid, username, calling, called)
         VALUES ('%g', '%s', '%u', '%{Calling-Station-Id}', '%{Called-Station-Id}')
  ```

- `StartQueryAlt=INSERT ...`
  Default: `N/A`
  Defines a SQL query used to insert a new call record to the database in case the `StartQuery` failed for some reason (the call already exists, for example). The syntax and parameters are the same as for `StartQuery`.

- `UpdateQuery=UPDATE ...`
  Default: `N/A`
  Defines a SQL query used to update a call record in the database with the current call state. It is used for connect and update accounting events. The syntax and parameters are the same as for `StartQuery`.

  Sample query string:

  ```
  UPDATE call SET duration = %d WHERE gkname = '%g' AND sessid = '%s'
  ```

- `StopQuery=UPDATE ...`
  Default: `N/A`
  Defines SQL query used to update a call record in the database when the call is finished (disconnected). The syntax and parameters are the same as for `StartQuery`.

  Sample query string:

  ```
  UPDATE call SET duration = %d, dtime = '%{disconnect-time}' WHERE gkname = '%g' AND sessid = '%s'
  ```

- `StopQueryAlt=INSERT ...`
  Default: `N/A`
  Defines a SQL query used to update call record in the database when the call is finished (disconnected) in case the regular `StopQuery` failed (because the call record does not yet exist, for example). The syntax and parameters are the same as for `StartQuery`.

  Sample query string:

  ```
  INSERT INTO call (gkname, sessid, username, calling, called, duration)
         VALUES ('%g', '%s', '%u', '%{Calling-Station-Id}', '%{Called-Station-Id}', %d)
  ```

- `AlertQuery=UPDATE ...`
  Default: `N/A`
  Defines SQL query used to update a call record in the database when the call is alerting.

- `RegisterQuery=INSERT ...`
  Default: `N/A`
  Defines SQL query used to update the database when an endpoint registers.

  Register and Unregister queries can use the following parameters:

  – `%g` - gatekeeper name
  – `%u` - H.323 ID of the registering party
  – %{`endpoint-ip`} - IP number of the endpoint
  – %{`endpoint-port`} - port number of the endpoint
  – %{`epid`} - the endpoint ID
  – %{`aliases`} - the comma deparated list of aliases the endpoint has registered with

- `UnregisterQuery=DELETE ...`
  Default: `N/A`
  Defines SQL query used to update the database when an endpoint unregisters.

- `OnQuery=UPDATE ...`
  Default: `N/A`
  Defines SQL query used to update a record in the database when the gatekeeper is starting.

- `OffQuery=UPDATE ...`
  Default: `N/A`
  Defines SQL query used to update a record in the database when the gatekeeper is stopping.

- `TimestampFormat=MySQL`
  Default: `N/A`
  Format of timestamp strings used in queries. If this setting is not specified, the global one from the main gatekeeper section is used.

- `MinPoolSize=5`
  Default: `1`
  Number of concurrent SQL connections in the pool. The first available connection in the pool is used to store accounting data.

9.5.1 A Sample MySQL Schema

The SQLAcct module is designed to adapt to whatever database structure you already have. You can define all queries so they fit your existing tables. Here is an example of what those tables might look like in MySQL and which you can use as a starting point.

Create a new database; here we use the name 'GNUGK':

```
create database GNUGK;
```

Then create a table in this database to store you accounting data; we call the table 'CDR'.

```
create table GNUGK.CDR (
        gatekeeper_name varchar(255),
        call_number int zerofill,
        call_duration mediumint unsigned zerofill,
                index duration_idx (call_duration),
        disconnect_cause smallint unsigned zerofill,
                index dcc_idx (disconnect_cause),
        acct_session_id varchar(255),
        h323_id varchar(255),
        gkip varchar(15),
        CallId varchar(255),
        ConfID varchar(255),
        setup_time datetime,
        connect_time datetime,
        disconnect_time datetime,
        caller_ip varchar(15),
                index srcip_idx (caller_ip),
        caller_port smallint unsigned zerofill,
        callee_ip varchar(15),
                index destip_idx (callee_ip),
        callee_port smallint unsigned zerofill,
        src_info varchar(255),
        dest_info varchar(255),
        Calling_Station_Id varchar(255),
        Called_Station_Id varchar(255),
                index dialednumber_idx (Called_Station_Id (20)),
        Dialed_Number varchar(255)
);
```

Then you need to create a username for accessing the data.

```
mysql> GRANT delete,insert,select,update ON GNUGK.* TO 'YourDesiredUsername'@'localhost' IDENTIFIED BY 'AP
mysql> flush privileges;
```

With this command you will permit access to the data only from the local server. If you need to access the data from any other computer then you have to set the proper security options.

For example, to permit access from the 192.168.1.0/24 network:

```
mysql> GRANT delete,insert,select,update ON GNUGK.* TO 'YourDesiredUsername'@'192.168.1.%' IDENTIFIED BY '
mysql> flush privileges;
```

Then you can add the following settings into your gnugk.ini file to insert and update the history of the calls into your database.

```
[Gatekeeper::Acct]
SQLAcct=optional;start,stop,update
FileAcct=sufficient;stop

[FileAcct]
DetailFile=Add your desire path here something like /var/log/cdr.log
StandardCDRFormat=0
CDRString=%g|%n|%d|%c|%s|%u|%{gkip}|%{CallId}|%{ConfId}|%{setup-time}|%{connect-time}|%{disconnect-time}|%
Rotate=daily
RotateTime=23:59

[SQLAcct]
Driver=MySQL
Database=GNUGK
Username=YourDesiredUsername
Password=APassword
StartQuery= insert into CDR (gatekeeper_name, call_number, call_duration, disconnect_cause, acct_session_i

StartQueryAlt= insert into CDR (gatekeeper_name, call_number, call_duration, disconnect_cause, acct_sessio

UpdateQuery= update CDR set call_duration=%d where gatekeeper_name='%g' and acct_session_id='%s'

StopQuery= update CDR set call_duration=%d, disconnect_cause=%c, disconnect_time='%{disconnect-time}' wher

StopQueryAlt= insert into CDR (gatekeeper_name, call_number, call_duration, disconnect_cause, acct_session

TimestampFormat=MySQL
```

9.6 Section [StatusAcct]

This accounting module sends all accounting information to the status port where it can be used to interface to external systems in real time.

You can use the 9.2 (common CDR parameters) to define what to include into your event strings.

- `StartEvent=CALL|Start|%{CallId}`
  Default: `CALL|Start|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event to display for a new call. The string is parametrized with the same variables as the other accounting modules (See 9.5 ([SQLAcct])).
- `StopEvent=CALL|Stop|%{CallId}`
  Default: `CALL|Stop|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event when a call is finished (disconnected). The syntax and parameters are the same as for `StartEvent`. This event is equivalent to the old status port CDR event, but more flexible.
- `UpdateEvent=CALL|Update|%{CallId}`
  Default: `CALL|Update|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines event used to update the current call state. The syntax and parameters are the same as for `StartEvent`.

- `AlertEvent=CALL|Alert|%{CallId}`
  Default: `CALL|Alert|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event when a call is alerting. The syntax and parameters are the same as for `StartEvent`.
- `ConnectEvent=CALL|Connect|%{CallId}`
  Default: `CALL|Connect|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event when a call is connected. The syntax and parameters are the same as for `StartEvent`.
- `RegisterEvent=EP|Register|%{endpoint-ip}`
  Default: `EP|Register|%{endpoint-ip}:%{endpoint-port}|%{aliases}`
  Defines the event when an endpoint registers. The syntax and parameters are the same as for `StartEvent`.
- `UnregisterEvent=EP|Unregister|%{endpoint-ip}`
  Default: `EP|Unregister|%{endpoint-ip}:%{endpoint-port}|%{aliases}`
  Defines the event when an endpoint registers. The syntax and parameters are the same as for `StartEvent`.
- `TimestampFormat=MySQL`
  Default: `N/A`
  Format of timestamp strings used in events. If this setting is not specified, the global one from the main gatekeeper section is used.

In addition to the CDR parameters, Register and Unregister events can use the following parameters:

- `%{endpoint-ip}` - IP number of the endpoint
- `%{endpoint-port}` - port number of the endpoint
- `%{epid}` - the endpoint ID
- `%{aliases}` - the comma deparated list of aliases the endpoint has registered with

9.7 Section [SyslogAcct]

This accounting module sends accounting information to the Unix syslog and is not available on Windows. The local syslog daemon will then route the messages according to its configuration, generally specified in /etc/syslog.conf.

You can use the 9.2 (common CDR parameters) to define what to include into your event strings.

- `SyslogFacility=LOG_LOCAL1`
  Default: `LOG_USER`
  Set the syslog facility to one of LOG_USER, LOG_DAEMON, LOG_AUTH, LOG_LOCAL0, LOG_LOCAL1, LOG_LOCAL2, LOG_LOCAL3, LOG_LOCAL4, LOG_LOCAL5, LOG_LOCAL6, LOG_LOCAL7.
- `SyslogLevel=LOG_NOTICE`
  Default: `LOG_INFO`
  Set the syslog level to LOG_EMERG, LOG_ALERT, LOG_CRIT, LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO or LOG_DEBUG.
- `StartEvent=CALL|Start|%{CallId}`
  Default: `CALL|Start|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event to display for a new call. The string is parametrized with the same variables as the other accounting modules (See 9.5 ([SQLAacct])).

- `StopEvent=CALL|Stop|%{CallId}`
  Default: `CALL|Stop|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event when a call is finished (disconnected). The syntax and parameters are the same as for `StartEvent`. This event is equivalent to the old status port CDR event, but more flexible.

- `UpdateEvent=CALL|Update|%{CallId}`
  Default: `CALL|Update|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines event used to update the current call state. The syntax and parameters are the same as for `StartEvent`.

- `ConnectEvent=CALL|Connect|%{CallId}`
  Default: `CALL|Connect|%{caller-ip}:%{caller-port}|%{callee-ip}:%{callee-port}|%{CallId}`
  Defines the event when a call is connected. The syntax and parameters are the same as for `StartEvent`.

- `TimestampFormat=MySQL`
  Default: `N/A`
  Format of timestamp strings used in events. If this setting is not specified, the global one from the main gatekeeper section is used.

10 Neighbor Configuration

10.1 Section [RasSrv::Neighbors]

If the destination of an ARQ is unknown, the gatekeeper sends LRQs to its neighbors to ask if they have the destination endpoint. A neighbor is selected if one of its prefixes matches the destination or it has the "`*`" prefix. More than one prefix may be specified. You can use special characters "`.`" to do wildcard matching and "`!`" to disable a specific prefix.

The gatekeeper will only reply to LRQs sent from neighbors defined in this section. If you specify an empty SendPrefixes entry, no LRQ will be sent to that neighbor, but the gatekeeper will accept LRQs from it.

The `password` field is used to authenticate LRQs from that neighbor. See section 8.1 ([Gatekeeper::Auth]) for details.

Whether a call is accepted from a neighbor also depends on the AcceptNeighborsCalls switch in the 5.1 ([RoutedMode]) section.

```
GKID="GnuGk" | "CiscoGk" | "ClarentGk" | "GlonetGk"
```

The gatekeeper types have the following characteristics:

- `Generic`
  For neighbors from any H.323 compliant vendor. (This setting is identical to 'GnuGk'.)
- `GnuGk`
  When in doubt, use the *GnuGk gatekeeper* type. This also activates H.460.23 / H.460.24.
- `CiscoGk`
  GnuGk will pretend to be a Cisco gatekeeper and send fake manufacturer data.
- `ClarentGk`
  Clarent gatekeeper can't decode nonStandardData in LRQs, so GnuGk will filter it out.
- `GlonetGk`
  Limited support for LRQ forwarding.

Example:

```
[RasSrv::Neighbors]
GK1=CiscoGk
GK2=GnuGk

[Neighbor::GK1]
GatekeeperIdentifier=GK1
Host=192.168.1.1
SendPrefixes=02
AcceptPrefixes=*
ForwardLRQ=always

[Neighbor::GK2]
GatekeeperIdentifier=GK2
Host=192.168.1.2
SendPrefixes=03,0048
AcceptPrefixes=0049,001
```

```
ForwardHopCount=2
ForwardLRQ=depends
```

The `[RasSrv::Neighbors]` section is only used to specify the gatekeeper type. The configuration for each neighbor is placed in a separate section.

10.2 Section [RasSrv::LRQFeatures]

Defines some features of LRQ and LCF.

- `NeighborTimeout=1`
  Default: `2`
  Timeout value in seconds to wait for responses from neighbors. If no neighbor responds before the timeout, the gatekeeper will reply with an ARJ to the endpoint sending the ARQ.

  This timout is applied to each retry (see below).

- `SendRetries=4`
  Default: `2`
  Number of tries to send LRQ to neighbors. If there is no response from neighbors after retries timeout, the gatekeeper will reply with a LRJ to the endpoint sending the LRQ.

- `ForwardHopCount=2`
  Default: `N/A`
  If the gatekeeper receives a LRQ that the destination is unknown it may forward this message to its neighbors.

  When the gatekeeper receives a LRQ and decides that the message should be forwarded on to another gatekeeper, it first decrements **hopCount** field of the LRQ. If **hopCount** has reached 0, the gatekeeper shall not forward the message. This option defines the number of gatekeepers through which a LRQ may propagate. Note that it only affects the sender of LRQ, not the forwarder. This setting can be overridden via the configuration section for a particular neighbor.

- `AcceptForwardedLRQ=1`
  Default: `1`
  Whether to accept an LRQ forwarded from neighbors. This setting can be overridden with configuration of a particular neighbor.

- `ForwardResponse=0`
  Default: `0`
  If the gatekeeper forwards a received LRQ message it can decide either to receive the LCF response or to let it travel back directly to the LRQ originator. Set this option to 1 if the gatekeeper needs to receive LCF messages for forwarded LRQs. This setting can be overridden with configuration of a particular neighbor.

- `ForwardLRQ=always | never | depends`
  Default: `depends`
  This settings determines whether the received LRQ should be forwarded or not. `always` forwards LRQ unconditionally, `never` blocks LRQ forwarding, `depends` tells the gatekeeper to forward LRQ only if its hop count is greater than 1. This setting can be overridden with configuration of a particular neighbor.

- `AcceptNonNeighborLRQ=1`
  Default: `0`
  Whether to accept a LRQ forwarded from parties not defined as Neighbors. This can be used with SRV routing policy to place calls to third party gatekeepers. This should be used in conjunction with a LRQ Authentication policy.

- `AcceptNonNeighborLCF=1`
  Default: `0`
  This setting disables matching of the LRQ responder's IP address and specified neighbor IP addresses in order to accept LCF message responses from any IP address. This has primary importance when a multiple level gatekeeper hierarchy is used without routed Q.931 signaling. As a minimal security, only LRQ/LCF sequence numbers will be checked accordingly. This feature is required by the national gatekeepers connected to the Global Dialing Scheme (GDS), see *http://www.vide.net/help/gdsintro.shtml* `<http://www.vide.net/help/gdsintro.shtml>` for more information. WARNING: Enabling receiving LCF from other than the LRQ destination IP is a significant security risk. Use this setting with extreme caution.

- `SendRIP=9000`
  Default: `0`
  Send a RequestInProgress (RIP) message with this delay value after receiving an LRQ. This switch can be used to extend the duration the caller will wait for an answer. No RIP is sent when the delay ist set to 0.

- `EnableLanguageRouting=1`
  Default: `0`
  Whether to compare users language settings in determining routing requests.

- `PingAlias=my-ping`
  Default: `gatekeeper-monitoring-check`
  Alias used for LRQ pinging. LRQs received with this alias will be processed a bit faster than reqular requests. This feature is also used by VCS in GDS dialing schema. Setting the alias alone will not turn on the sending of pings.

- `SendLRQPing=1`
  Default: `0`
  When enabled, GnuGk will periodically ping all its neighbors with a LRQ. If the neighbor doesn't respond with a LRQ or LRJ withing the NeighborTimeout, the neighbor is disabled and won't be used for routing until the next LRQ ping succeeds. Skipping disabled neighbors can speed up routing in some configurations.

  You can configure the alias in the LRQ with the PingAlias switch.

- `LRQPingInterval=30`
  Default: `60`
  Interval to be used for sending LRQ pings.

10.3 Section [Neighbor::...]

Sections starting with `[Neighbor::` are specific for one neighbor. If you define a [Neighbor::...] section, the default values of all settings in 10.2 ([RasSrv::LRQFeatures]) will be applied to this neighbor. You may override the global defaults through configuration options in each neighbor-specific section.

- `GatekeeperIdentifier=GKID`
  Default: `N/A`

Gatekeeper identifier for this neighbor. If this option is not specified, the identifier is taken from the second part of the `Neighbor::` section name.

- `Host=192.168.1.1`
  Default: `N/A`
  An IP address for this neighbor.

- `Password=secret`
  Default: `N/A`
  A password to be used to validate crypto tokens received in incoming LRQs and SCIs. Encrypted if Keyfilled= is set, plain text otherwise.

- `AuthUser=Foo`
  Default: `GKID`
  The user name to be used to validate crypto tokens received in incoming LRQs and SCIs. The default value is the gatekeeper identifier for this neighbor (see above).

- `Dynamic=0`
  Default: `0`
  1 means that the IP address for this neighbor can change.

- `SendPrefixes=004,002:=1,001:=2`
  Default: `N/A`
  A list of prefixes that this neighbor expects to receive LRQs for. If '*' is specified, LRQs will always be sent to this neighbor. A priority can be given to each prefix for each neighbor (using := syntax), so in case of multiple LCF received from multiple neighbor, the one with the highest priority will be selected to route the call. One can also direct the gatekeeper to send LRQ to this neighbor based on an alias type:
  SendPrefixes=h323_ID,dialedDigits,001

- `SendIPs=192.168.0.0/16,172.16.0.0/12`
  Default: `N/A`
  Send calls dialed by IP to this neighbor. You can specify a list of networks with optional netmask. You can also put a `!` in front of the network for negation. Special values are "*" to send all IP calls, "private" to send all IPv4 private networks and "public" to send all public IPv4 adresses to this neighbor. If one of the networks matches the dialed IP, the neighbor is selected.

  If the call comes from a registered endpoint, this endpoint must support canMapAlias for ARQs.

- `SendAliases=4526354,2000-2010,frank`
  Default: `N/A`
  A list of specific aliases this neighbor expects to receive LRQs for. For E.164 numbers, ranges can be specified.

- `AcceptPrefixes=*`
  Default: `*`
  A list of prefixes that GnuGk will accept in LRQs received from this neighbor. If '*' is specified, all LRQs will be accepted from this neighbor. One can also direct the gatekeeper to accept LRQ from this neighbor based on an alias type:
  AcceptPrefixes=dialedDigits

- `ForwardHopCount=2`
  Default: `N/A`

If the gatekeeper receives an LRQ that the destination is either unknown, it may forward this message to its neighbors. When the gatekeeper receives an LRQ and decides that the message should be forwarded on to another gatekeeper, it first decrements **hopCount** field of the LRQ. If **hopCount** has reached 0, the gatekeeper shall not forward the message. This options defines the number of gatekeepers through which an LRQ may propagate. Note it only affects the sender of LRQ, not the forwarder.

- `AcceptForwardedLRQ=1`
  Default: `1`
  Whether to accept an LRQ forwarded from this neighbor.

- `ForwardResponse=0`
  Default: `0`
  If the gatekeeper forwards received LRQ message it can decide either to receive the LCF response or to let it travel back directly to the LRQ originator. Set this option to `"1"` if the gatekeeper needs to receive LCF messages for forwarded LRQs.

- `ForwardLRQ=always | never | depends`
  Default: `depends`
  This settings determines whether the received LRQ should be forwarded or not. `always` forwards LRQ unconditionally, `never` blocks LRQ forwarding, `depends` tells the gatekeeper to forward LRQ only if its hop count is greater than 1.

- `H46018Client=1`
  Default: `0`
  Enable H.460.18 keep-alive messages to this neighbor and act as a traversal client.

- `H46018Server=1`
  Default: `0`
  Act as a traversal server for another gatekeeper which is configured as traversal client.

  No two neighbors for which we are acting as traversal server should have the same AuthUser name. Since the IP of the traversal client can be unknown or changing, the user name is used to update the IP for this neighbor.

- `SendPassword=secret`
  Default: `N/A`
  The password to send to the neighbor (right now only used for H.460.18 SCI). Encrypted if Keyfilled= is set, plain text otherwise.

- `SendAuthUser=Foo`
  Default: `own GK-ID`
  The user name (gatekeeprID) to be used to send crypto tokens to this neighbor (right now only used for H.460.18 SCI). The default value is this gatekeeper's ID.

- `UseTLS=1`
  Default: `0`
  Use TLS (transport layer security) with this neighbor. See also 12.11 ([TLS] section).

- `SendLRQPing=1`
  Default: `0`
  Enable LRQ ping only for this neighbor.

10.4 Configuring a Traversal Zone with GnuGk as Traversal Server

To configure a traversal zone with a Tandberg VCS, add a Zone of type "Traversal client" in the VCS.

The user name and password configured in the VCS should be set as AuthUser= and Password= in the [Neighbor::..] section. The password must be encoded with the addpasswd tool if the Keyfilled= switch is used, otherwise it is entered as plain text in the config. Please note that for any password authentication to work, both systems must have accurate and synchronized time, so it is strongly recommended that you configure NTP.

Enable H.323 in the VCS settings, set the Protocol to H.460.18 (not Assent) and the port to 1719.

Add the IP of your GnuGk server as the Peer 1 address in the VCS.

Enable H.460.18 in your GnuGk config with EnableH46018=1 in the [RoutedMode] section. Set H46018Client=0 and H46018Server=1 in the [Neighbor::..] section. If H.460.18 is globally enabled, GnuGk will automatically detect that a neighbor is acting like a H.460.18 traversal zone client and it needs to act as a traversal server. But since traversal clients may come from unknown or changing IPs, setting the H46018Server flag explicitly allows GnuGk to update the client's IP on the first keepAlive SCI message.

Example:

```
[RoutedMode]
EnableH46018=1

[RasSrv::Neighbors]
VCSClient=Generic

[Neighbor::VCSClient]
GatekeeperIdentifier=FooVCS
Host=192.168.1.1
SendPrefixes=02
AcceptPrefixes=*
H46018Client=0
H46018Server=1
AuthUser=clientuser
Password=clientpw
```

10.5 Configuring a Traversal Zone with GnuGk as Traversal Client

To configure a traversal zone with a Tandberg VCS, add a Zone of type "Traversal server" in the VCS. When functioning as a traversal server, the VCS usually uses a different port, so make sure you add the port to the Host switch.

Enable H.323 in the VCS settings, set the Protocol to H.460.18 (not Assent) and select a port (you can't use 1719!). You must specify this port in your GnuGk config for this neighbor. Set a username and password in the VCS and put them into SendAuthUser= and SendPaswword= in your GnuGk config.

In the GnuGk config, set EnableH46018=1 in [RoutedMode] and set H46018Client=1 in the [Neighbor::..] section.

Please note that for any password authentication to work, both systems must have accurate and synchronized time, so it is strongly recommended that you configure NTP.

Example:

```
[RoutedMode]
EnableH46018=1
```

```
[RasSrv::Neighbors]
VCSServer=Generic

[Neighbor::VCSServer]
;from unknown IP
Host=211.211.10.10:9004
SendPrefixes=*
AcceptPrefixes=*
H46018Client=1
H46018Server=0
SendAuthUser=serveruser
SendPassword=serverpw
```

11 Per-Endpoint Configuration

In addition to the standard configuration file options, per-endpoint configuration settings can be specified in the *GnuGk* config file. The syntax is as follows:

11.1 Section [EP::...]

```
[EP::ALIAS]
Key Name=Value String
```

`ALIAS` should be replaced with the actual alias for the endpoint the settings should apply to. If your endpoint has multiple aliases, you can pick one of them. GnuGk will look through all aliases and use the first [EP:..] section it finds.

Currently, the following options are recognized:

- `Capacity=10`
  Default: `-1`
  Call capacity for an endpoint. No more than `Capacity` concurrent calls will be sent to this endpoint. In case of gateways, if more than one gateway matches a dialed number, a call will be sent to the first available gateway which has available capacity.
- `PrefixCapacities=^0049:=10,^(0044|0045):=20`
  Default: `N/A`
  Limit the capacity for certain prefixes. Regular expressions can be used to specify the prefix and specify a combined capacity for a group of prefixes. For a gateway to be considered available a.) the prefix must have capacity left and b.) the total gateway capacity (see above) must not be exceeded.
- `ForceGateway=1`
  Default: `0`
  If you endpoint doesn't register as a gateway or MCU, you can't assign gateway prefixes. Use this switch to treat an endpoint as a gateway regardless of how it regsiters.
- `GatewayPriority=1`
  Default: `1`
  Applicable only to gateways. Allows priority based routing when more than one gateway matches a dialed number. Lower values indicate a higher gateway priority. A call is routed to the first available gateway (that has available capacity) with the highest priority (the lowest `GatewayPriority` values). In case the gateway priority contradicts prefix priority (see section 7.2 ([RasSrv::GWPrefixes])) for details), prefix priority will take precedence.
- `GatewayPrefixes=0048,0049:=2,0044`
  Default: `N/A`
  Additional prefixes for this gateway. Applies only to gateways. Special characters `.` and `!` can be used to match any digit or to disable the prefix. You may use the := syntax to set a prefix priority in the same manner as in 7.2 ([RasSrv::GWPrefixes]) section. If no priority is explicitly configured for a prefix, then the gateway priority is used.
- `AddNumbers=4212,5650-5630,6000`
  Default: `N/A`
  Add E.164 numbers to this endpoint. The new aliases can either be specified as a list of numbers or as number ranges.

- `CalledTypeOfNumber=1`
Default: `N/A`
Sets Called-Party-Number type of number to the specified value for calls sent to this endpoint (0 - UnknownType, 1 - InternationalType, 2 - NationalType, 3 - NetworkSpecificType, 4 - SubscriberType, 6 - AbbreviatedType, 7 - ReservedType).

- `CallingTypeOfNumber=1`
Default: `N/A`
Sets Calling-Party-Number type of number to the specified value for calls sent to this endpoint (0 - UnknownType, 1 - InternationalType, 2 - NationalType, 3 - NetworkSpecificType, 4 - SubscriberType, 6 - AbbreviatedType, 7 - ReservedType).

- `CalledPlanOfNumber=1`
Default: `N/A`
Sets Called-Numbering-Plan of number to the specified value for calls sent to this endpoint (0 - UnknownType, 1 - ISDN, 3 - X.121 numbering, 4 - Telex, 8 - National standard, 9 - private numbering).

- `CallingPlanOfNumber=1`
Default: `N/A`
Sets Calling-Numbering-Plan of number to the specified value for calls sent to this endpoint (0 - UnknownType, 1 - ISDN, 3 - X.121 numbering, 4 - Telex, 8 - National standard, 9 - private numbering).

- `Proxy=1`
Default: `0`
Enables/disables proxying calls sent to this endpoint (0 - do not change global proxy settings, 1 - force proxy mode, 2 - disable proxy mode).

- `TranslateReceivedQ931Cause=17:=34`
Default: `N/A`
Translate received cause codes in ReleaseComplete messages from this endpoint. In the above example code 17 (User busy) will be translated into cause code 34 (No circuit/channel available).

- `TranslateSentQ931Cause=21:=34,27:=34`
Default: `N/A`
Translate cause codes in ReleaseComplete messages sent out to this endpoint. In the above example code 21 and 27 will be translated into cause code 34, because this particular gateway might deal with error code 34 better than with others.

- `DisableH46017=1`
Default: `0`
Disable H.460.17 for this endpoint.

- `DisableH46018=1`
Default: `0`
Disable H.460.18/.19 for this endpoint.

- `MaxBandwidth=81920`
Default: `-1`
Maximum bandwidth this endpoint may use in units of 100 bits per second.

  This includes inbound and outbound bandwidth, so for symmetrical calls this should be doubled.

  Common bandwidth settings and their GnuGk equivalents:
384K = 7680
512K = 10240
768K = 15360

1024K = 20480
1920K = 38400
2048K = 40960
4096K = 81920

If you do not wish to restrict bandwidth, then use "-1".

- `AdditionalDestinationAlias=H323-TRUNK`
  Default: `n/a`
  Add an additional alias to all calls going to this endpoint, if its not already present. This can be used to add the trunk ID required by Swxy version 6 and up.

- `UseTLS=1`
  Default: `0`
  Use TLS (transport layer security) when calling this endpoint. See also 12.11 ([TLS] section). If the endpoint is able to signal it's capability to use TLS by H.460.22, then there is no need to explicitely configure this.

- `DisableCallCreditCapabilities`
  Default: `0`
  If you have an endpoint that signals call credit capabilities, but crashes when they are used, you can use this switch to disable the feature for this endpoint.

- `AddCallingPartyToSourceAddress=1`
  Default: `0`
  Tell GnuGk to add the number from calling party IE to the list of source addresses in the Setup message.

Example how to attach an [EP::..] section to an endpoint:

```
[RasSrv::PermanentEndpoints]
192.168.1.1=gw1;48
192.168.1.2=gw2;48,!4850,!4860,!4869,!4888

[EP::gw1]
Capacity=60
GatewayPriority=1

[EP::gw2]
Capacity=30
GatewayPriority=2
```

In this example, calls will be sent to the gateway `gw1` until its capacity is fully utilized (60 concurrent calls) and then to the gateway `gw2`.

12 Advanced Configuration

12.1 Section [CallTable]

- `GenerateNBCDR=0`
  Default: `1`
  Generate CDRs for calls from neighbor zones on the status port. The IP and endpoint ID of the calling party is printed as empty. This is usually used for debugging purposes. The accounting modules will always get CDR data for all calls.

- `GenerateUCCDR=0`
  Default: `0`
  Generate CDRs for calls that are unconnected. This is usually used for debugging purposes. Note that a call is considered unconnected only if the gatekeeper uses routed mode and a Q.931 "Connect" message is not received by the gatekeeper. In direct mode, a call is always considered connected.

- `DefaultCallDurationLimit=3600`
  Default: `0`
  Default maximum call duration limit (seconds). Set it to `0` to disable this feature and not limit call duration.

- `AcctUpdateInterval=60`
  Default: `0`
  A time interval (seconds) for accounting updates to be logged for each call in progress. The exact details of the accounting updates depend on accounting logger modules selected (see section 9.1 ([Gatekeeper::Acct])). In general, the accounting update is to provide back-end services with incrementing call duration for connected calls. The default value "0" disables accounting updates. Please note that setting this to a short interval may decrease gatekeeper performance.

- `TimestampFormat=Cisco`
  Default: `RFC822`
  Format of timestamp strings printed inside CDRs. You can use the same list of formats as specified in the 4.5 ([Gatekeeper::Main]) section.

- `IRRFrequency=60`
  Default: `120`
  Set the irrFrequency in ACF messages. 0 turns it off.

- `IRRCheck=TRUE`
  Default: `FALSE`
  Check if both endpoints in a call send the requested IRRs. A call will be terminated if one of the endpoints do not send an IRR after 2 * irrFrequency.

- `SingleFailoverCDR=FALSE`
  Default: `TRUE`
  When failover is active, more than one gateway may be tried to establish a call. This switch defines if one or multiple CDRs are generated for such a call.

- `DisabledCodecs=g711Alaw64k;g711Ulaw64k;h263VideoCapability;genericVideoCapability;basicString;`
  Default: `N/A`
  Filter out certain codecs. Calls must be H.245 routed or proxied for codec filtering to work. You can also filter out whole capability sets, eg. "receiveVideoCapability", "receiveAndTransmitDataApplicationCapability" of "nonStandard". This setting can be overridden on a per-call basis by using the Radius attribute 'disable-codec'.

12.2 Section [H225toQ931]

When converting between H.225 reasons and Q.931 cause codes, GnuGk uses a conversion table. Using this section you can change this mapping.

```
   [H225toQ931]
;0=34 # noBandwidth
;1=47 # gatekeeperResources
2=34 # unreachableDestination => NoCircuitChannelAvailable (default 3)
;3=16 # destinationRejection
;4=88 # invalidRevision
;5=111 # noPermission
;6=38 # unreachableGatekeeper
;7=42 # gatewayResources
;8=28 # badFormatAddress
;9=41 # adaptiveBusy
;10=17 # inConf
;11=31 # undefinedReason
;12=16 # facilityCallDeflection
;13=31 # securityDenied
14=34 # calledPartyNotRegistered => NoCircuitChannelAvailable (default 20)
;15=31 # callerNotRegistered
;16=47 # newConnectionNeeded
;17=127 # nonStandardReason
;18=31 # replaceWithConferenceInvite
;19=31 # genericDataReason
;20=31 # neededFeatureNotSupported
;21=127 # tunnelledSignallingRejected
```

12.3 Section [GkQoSMonitor]

Use H.460.9 to collect Quality of Service information from endpoints. Endpoints must support H.460.9 for this service to function.

- `Enable=1`
  Default: `0`
  Defines whether to enable or disable the feature. If enabled, this function will respond to supported-Feature requests from clients so clients know to send QoS statistics to the gatekeeper.

- `CallEndOnly=0`
  Default: `1`
  Defines whether to collect the information via IRR messages or only collect QoS information at the end of a call.

- `DetailFile=qos.txt`
  Default: `N/A`
  Define the output file for QoS logs. If a file is not defined the QoS information is output as an item in the Trace File at trace level 4.

12.4 Section [GkQoSMonitor::SQL]

This section allows you to store QoS information in a database. You can use the same database parameters as defined in 8.4 ([SQLPasswordAuth]).

- `Query=INSERT ...`
  Default: `N/A`
  Defines the SQL query used to store the QoS information.

  The following parameters are defined:

  - `%g` - gatekeeper ID
  - `%{ConfId}` - conference ID
  - `%{session}` - session
  - `%{caller-ip}` - caller IP
  - `%{caller-port}` - caller port
  - `%{caller-nat}` - is caller NATted (0 or 1)
  - `%{callee-ip}` - caller IP
  - `%{callee-port}` - caller port
  - `%{avgdelay}` - average delay
  - `%{packetloss}` - packet loss
  - `%{packetloss-percent}` - packet loss percentage
  - `%{avgjitter}` - average jitter
  - `%{bandwidth}` - bandwidth (in units of 100 bits per second)
  - `%t` - timestamp

  Sample query string:

  ```
  INSERT INTO qos SET caller_ip="%{caller-ip}", bandwidth="%{bandwidth}, timestamp=%t
  ```

12.5 Section [Endpoint]

The gatekeeper can function as an endpoint by registering with another gatekeeper, allowing you to build gatekeeper hierarchies. This section defines the endpoint features for the gatekeeper.

- `Gatekeeper=10.0.1.1`
  Default: `no`
  Define a parent gatekeeper for *GnuGk* to register with. When a call in the routing process reaches the 'parent' routing policy, it will route all calls to this gatekeeper. If you set this to `auto`, GnuGk will send an IPv4 broadcast GRQ.

  Make sure you don't register with yourself, the results can be very confusing.

- `Type=Gateway`
  Default: `Gateway`
  Define the terminal type GnuGk will use when it registers. Valid options are `Gateway` or `Terminal`.

- `Vendor=Cisco | GnuGk | Generic`
  Default: `GnuGk`
  Choose parent gatekeeper type to enable vendor specific extensions. This setting can also be configured by specifying the triplet of T.35 IDs (<country>,<manufacturer>,<extension>) for the vendor to set the vendor ID without enabling any extensions.

Example:

```
Vendor=GnuGk
; Aculab ID
Vendor=222,173,0
```

- `HideGk=1`
  Default: `0`
  Hide the fact that a gatekeeper is registering, eg. don't set gatekeeper flag in terminal type.

- `ProductId=Any Product Name`
  Default: `GnuGk's name`
  This setting will allow you to configure the "product ID" string used in the registration request. The primary use of this setting would be to solve interoperability issues.

- `ProductVersion=1.2.3`
  Default: `GnuGk version`
  This setting allows you to configure the product version string in the registration request in order to solve interoperability issues.

- `H323ID=ProxyGK`
  Default: `<Name>`
  Specify the H.323 ID aliases for the endpoint. Multiple aliases can be separated with a comma.

- `E164=18888600000,18888700000`
  Default: `N/A`
  Define the E.164 (dialedDigits) aliases for the endpoint. Multiple aliases can be separated with a comma.

- `Password=123456`
  Default: `N/A`
  Specify a password to be sent to the parent gatekeeper.

  All RAS requests will contain the password in the **cryptoTokens** field (MD5 & HMAC-SHA1-96) and the **tokens** field (CAT). To send RAS requests without the **cryptoTokens** and **tokens** fields, configure an empty password. If `EncryptAllPasswords` is enabled, or a `KeyFilled` variable is defined in this section, the password is in encrypted form and should be created using the `addpasswd` utility.

  The password will be used in LRQs sent to neighbor gatekeepers.

- `Prefix=188886,188887`
  Default: `N/A`
  Register the specified prefixes with the parent gatekeeper. Only takes effect when the Type is `Gateway`.

- `TimeToLive=900`
  Default: `60`
  Suggest a time-to-live value (in seconds) for the registration. Note that the real time-to-live timer is assigned by the parent gatekeeper in the RCF is sends to us in response to our RRQ.

- `RRQRetryInterval=10`
  Default: `3`
  Define a retry interval in seconds for resending an RRQ if no response is received from the parent gatekeeper. This interval is doubled with each failure, up to a maximum RRQRetryInterval * 128 timeout.

- `UnregisterOnReload=1`
  Default: `0`

Defines whether the child gatekeeper unregisters and re-registers with its parent after receiving a Reload command from the status port.

- `NATRetryInterval=60`
  Default: `60`
  How long to wait before trying to reconnect TCP NAT signaling socket (seconds). This can happen when either the connection cannot be established or it has been broken.

- `NATKeepaliveInterval=86400`
  Default: `86400`
  Define how often the TCP NAT signaling connection with a parent gatekeeper is refreshed. As NAT boxes usually keep TCP mappings for a certain duration, it's strongly suggested to set this to a value slightly shorter than the NAT box mapping timeout. Refreshing is done by sending a special Q.931 IncomingCallProceeding message. If your NAT performs TCP port translation, you may need to set it to a value as short as 60 seconds.

- `Discovery=0`
  Default: `1`
  Configures GnuGk to attempt to discover the parent gatekeeper by first sending a GRQ.

- `UseAlternateGK=0`
  Default: `1`

  Enable alternate gatekeepers feature. If GRJ/GCF/RCF messages received from a parent gatekeeper contain a list of alternate gatekeepers, this information is stored and can be used to re-register with another gatekeeper in case of failure. If you don't want to use this feature, set this variable to `0`.

- `GatekeeperIdentifier=ParentGK`
  Default: `Not set`
  Define this parameter if you only want to accept parent gatekeepers that match this gatekeeper identifier. Useful with GRQ discovery and can prevent an accidental gatekeeper match. Do not set this variable if you do not care about gatekeeper identifiers or you use alternate gatekeepers that can have different gatekeeper identifiers.

- `EndpointIdentifier=ChildGK`
  Default: `Not set`
  Set this if you want to use a specific endpoint identifier for this child gatekeeper. If this option is not set (default), the identifier is assigned by a parent gatekeeper in a GCF/RCF message.

- `ForwardDestIp=0`
  Default: `1`
  Forward the destCallSignalAddress in ARQs to the parent gatekeeper.

- `EnableAdditiveRegistration=1`
  Default: `0`
  Whether the child gatekeeper supports passing registration information to the parent. Use this if you wish to manage registrations including authentication in the parent gatekeeper and not the child.

- `EnableH46018=1`
  Default: `0`
  Whether the child offers H.460.18/.19 support to the parent.

- `EnableH46023=1`
  Default: `0`
  Whether the child offers H.460.23/.24 support to the parent.

- `UseTLS=1`
  Default: `0`
  Use TLS (transport layer security) with this gatekeeper. See also 12.11 ([TLS] section).

12.6 Section [CTI::Agents]

This section allows the configuration of a so-called virtual queue to allow inbound call distribution by an external application via the status port. A virtual queue has a H.323 alias that can be called like an endpoint or it can answer to a set of aliases.

Once a call arrives on the virtual queue, the gatekeeper signals a RouteRequest on the status port and waits for an external application to respond with either a RouteReject (which will cause the call to be rejected) or with RouteToAlias/RouteToGateway which leads to the destination being rewritten so the call will be routed to the alias (eg. call center agent) specified by the external application.

If no answer is received after a timeout period, the call is terminated.

You can specify virtual queues in three ways:

- `exact alias name` - a list of aliases is given. If a request destination alias matches one these names, the virtual queue is activated.
- `prefix` - a list of prefixes is given. If a request destination alias starts with one these prefixes, the virtual queue is activated.
- `regular expression` - a regular expression is given. If a request destination alias matches the expression, the virtual queue is activated.

To apply the virtual queue to all calls, specify a regular expression that matches everything, see the example below.

See the monitoring section for details on the messages and responses.

- `VirtualQueueAliases`
  Default: `none`
  This defines a list of H.323 aliases for the virtual queues (used with the vqueue RoutingPolicy).

  Example:

  ```
  VirtualQueueAliases=sales,support
  ```

- `VirtualQueuePrefixes`
  Default: `none`
  This defines a list of prefixes for the virtual queues (used with the vqueue RoutingPolicy).

  Example:

  ```
  VirtualQueuePrefixes=001215,1215
  ```

- `VirtualQueueRegex`
  Default: `none`
  This defines a regular expression for the virtual queues (used with the vqueue RoutingPolicy).

  Example (numbers starting with 001215 or 1215):

  ```
  VirtualQueueRegex=^(001|1)215[0-9]*$
  ```

  Example to match all calls:

```
VirtualQueueRegex=^.*$
```

- `RequestTimeout`
  Default: `10`
  Timeout in seconds for the external application to answer the RouteRequest. If no answer is received during this time the call will be rejected.

12.7 Section [CTI::MakeCall]

This section contains the settings for the status port command 13.2 (MakeCall).

- `EndpointAlias=DialOut`
  Default: `InternalMakeCallEP`
  This defines the endpoint alias for the pseudo endpoint used to dial.
- `TransferMethod=H.450.2`
  Default: `FacilityForward`
  Set the method to transfer the call from the pseudo endpoint to the actual destination. Possible values are: FacilityForward, FacilityRouteCallToMC and H.450.2.
- `UseH450=1`
  Default: `0`
  Use a H.450.2 transfer instead of a Facility message to transfer the call from the pseudo endpoint to the actual destination. **Deprecated: Use TransferMethod= instead.**
- `Gatekeeper=192.168.1.2`
  Default: `127.0.0.1`
  Gatekeeper IP for the pseudo endpoint to register with.
- `Interface=192.168.1.1:1730`
  Default: `*:1722`
  Interface and port to use for the pseudo endpoint.
- `DisableFastStart=1`
  Default: `0`
  Disable FastStart for the pseudo endpoint.
- `DisableH245Tunneling=1`
  Default: `0`
  Disable H.245 tunneling for the pseudo endpoint.

12.8 Section [SQLConfig]

Load gatekeeper settings from a SQL database (in addition to settings read from the config file). A generic `ConfigQuery` can be used to read almost all setting from the database and/or one of `[RasSrv::RewriteE164]`, `[RasSrv::PermanentEndpoints]`, `[RasSrv::Neighbors]`, `[RasSrv::GWPrefixes]` queries can be used to load particular settings. Entries read from the SQL database take precedence over settings found in the config file.

Use the 4.3 (common database configuration options) to define your database connection for this module.

- `ConfigQuery=SELECT ...`
  Default: `N/A`

Define a SQL query used to read gatekeeper settings from the database. The query is parameterized - that means parameter replacement occurs before the query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit in a string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `ConfigQuery` only one parameter is defined:

- `%1` - the gatekeeper identifier

It is expected that the query returns zero or more rows of data, with each row consisting of **three** columns:

- `column at index 0` - config section name
- `column at index 1` - config key (option name)
- `column at index 2` - config value (option value)

Sample query strings:

```
ConfigQuery=SELECT secname, seckey, secval FROM sqlconfig WHERE gk = '%1'
ConfigQuery=SELECT 'RasSrv::RRQAuth', alias, rule FROM rrqauth WHERE gk = '%1'
```

- `RewriteE164Query=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve rewrite rules from the database for the `[RasSrv::RewriteE164]` section. The query is parameterized - that means parameter replacement occurs before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `RewriteE164Query` only one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of two columns:

  - `column at index 0` - rewrite rule key
  - `column at index 1` - rewrite rule value

  Sample query strings:

  ```
  RewriteE164Query=SELECT rkey, rvalue FROM rewriterule WHERE gk = '%1'
  ```

- `RewriteAliasQuery=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve rewrite rules from the database for the `[RasSrv::RewriteAlias]` section. The query is parameterized - that means parameter replacement occurs before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `RewriteAliasQuery` only one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of two columns:

  - `column at index 0` - rewrite rule key
  - `column at index 1` - rewrite rule value

  Sample query strings:

  ```
  RewriteAliasQuery=SELECT rkey, rvalue FROM assignedalias WHERE gk = '%1'
  ```

- `AssignedAliasQuery=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve rewrite rules from the database for the `[RasSrv::AssignedAlias]` section. The query is parameterized - that means parameter replacement occurs before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `AssignedAliasQuery` only one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of two columns:

  - `column at index 0` - rewrite rule key
  - `column at index 1` - rewrite rule value

  Sample query strings:

  ```
  AssignedAliasQuery=SELECT rkey, rvalue FROM assignedalias WHERE gk = '%1'
  ```

- `NeighborsQuery=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve neighbor entries from the database for the `[RasSrv::Neighbors]` section. The query is parameterized - that means parameter replacement occurs before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `NeighborsQuery` one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of six columns:

  - `column at index 0` - neighbor name (identifier)
  - `column at index 1` - neighbor IP address
  - `column at index 2` - neighbor port number
  - `column at index 3` - optional prefixes (comma separated)
  - `column at index 4` - optional password
  - `column at index 5` - optional dynamic IP flag

  Sample query strings:

  ```
  NeighborsQuery=SELECT nid, nip, nport, npfx, NULL, 0 FROM neighbor WHERE gk = '%1'
  ```

- `NeighborsQuery2=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve neighbor entries for new style format from the database for the `[RasSrv::Neighbors]` section. The query is parameterized - that means parameter replacement occurs before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `NeighborsQuery` one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of six columns:

  - `column at index 0` - neighbor name (identifier)

  - `column at index 1` - neighbor type (GnuGk, Cisco, Generic)
  - `column at index 2` - neighbor Host (IP [and port])
  - `column at index 3` - optional SendPrefixes (comma separated)
  - `column at index 4` - optional AcceptPrefixes (comma separated)
  - `column at index 5` - optional ForwardHopCount
  - `column at index 6` - optional AcceptForwardedLRQ
  - `column at index 7` - optional ForwardResponse
  - `column at index 8` - optional H46018Type (0-none 1-server 2-client)
  - `column at index 9` - optional SendAuthUser (H46018)
  - `column at index 10` - optional SendPassword (H46018)

  Sample query strings:

  ```
  NeighborsQuery2=SELECT id, gtype, host, sendPrefix, recvPrefix FROM neighbor WHERE gk ='%1';
  ```

- `PermanentEndpointsQuery=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve permanent endpoints from the database for the `[RasSrv::PermanentEndpoints]` section. The query is parameterized - that means parameter replacement occurs before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `PermanentEndpointsQuery` only one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of 3 or 5 columns:

  - `column at index 0` - permanent endpoint IP address
  - `column at index 1` - permanent endpoint port number
  - `column at index 2` - permanent endpoint alias
  - `column at index 3` - optional: permanent endpoint prefixes and priorities (comma separated)
  - `column at index 4` - optional: permanent endpoint vendor
  - `column at index 5` - optional: permanent endpoint product

  Sample query strings:

  ```
  PermanentEndpointsQuery=SELECT peip, 1720, pealias, NULL FROM permanentep WHERE gk = '%1'
  ```

- `GWPrefixesQuery=SELECT ...`
  Default: `N/A`
  Define a SQL query used to retrieve gateway prefixes from the database for the `[RasSrv::GWPrefixes]` section. The query is parameterized - that means parameter replacement is made before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `GWPrefixesQuery` only one parameter is defined:

  - `%1` - the gatekeeper identifier

  It is expected that the query returns zero or more rows of data, with each row consisting of two columns:

- `column at index 0` - gateway alias
- `column at index 1` - gateway prefixes (comma separated)

Sample query strings:

```
GWPrefixesQuery=SELECT gwalias, gwpfx FROM gwprefix WHERE gk = '%1'
```

12.9 Section [PortNotifications]

GnuGk can execute a system command whenever it opens a new port for listening. For example, this can be used to automatically update the firewall configuration.

The following placeholder are available:

- `%p` - protocol ("udp" or "tcp")
- `%n` - port number
- `%i` - IP

By configuring a command to run for some types of ports, but not for others, you can easily choose which ports to handle and which to ignore.

- `Q931PortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `Q931PortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `H245PortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `H245PortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `RTPPortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `RTPPortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `T120PortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `T120PortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `RASPortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `RASPortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `StatusPortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `StatusPortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `RadiusPortOpen=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

- `RadiusPortClose=/usr/local/bin/ports.sh %p %n %i`
  Default: `none`

Example:

```
[PortNotifications]
Q931PortOpen=/usr/local/bin/ports.sh %p %n %i
Q931PortClose=/usr/local/bin/ports.sh %p %n %i
H245PortOpen=/usr/local/bin/ports.sh %p %n %i
H245PortClose=/usr/local/bin/ports.sh %p %n %i
RTPPortOpen=/usr/local/bin/ports.sh %p %n %i
RTPPortClose=/usr/local/bin/ports.sh %p %n %i
RASPortOpen=/usr/local/bin/ports.sh %p %n %i
RASPortClose=/usr/local/bin/ports.sh %p %n %i
T120PortOpen=/usr/local/bin/ports.sh %p %n %i
T120PortClose=/usr/local/bin/ports.sh %p %n %i
```

12.10 Section [SNMP]

The Simple Network Management Protocol (SNMP) lets you monitor the GNU Gatekeeper during operation and allows you to configure a destination system for notifications ("traps") when an error occurs.

- `EnableSNMP=1`
  Default: `0`
  Enable SNMP support. GnuGk must be compiled with SNMP support.

- `Implementation=PTLib`
  Default: `Net-SNMP`
  Select the SNMP implementation to use. Possible values are `Net-SNMP`, `PTLib` and `Windows`.

- `EnableWarningTraps=1`
  Default: `0`
  Also throw traps with warnings. If you enable warning traps, make sure you have good trap filtering in place in your management application. Many warnings can be generated in harmles situations where GnuGk can't decide if its a real error or not.

If the GnuGk configuration file enables SNMP, and the Net-SNMP implementation is selected, then GnuGk will register as an AgentX sub-agent with the Net-SNMP daemon. By default GnuGk will connect to the localhost IP address (127.0.0.1) and TCP port 705. Registering as a sub-agent allows you to continue querying Net-SNMP about the general health of the server and adds a GnuGk specific object.

All GET / SET requests and SNMP traps are performed by the Net-SNMP daemon, so configuration of access control, trap destination and SNMP version information must be done in the Net-SNMP .conf file.

If PTLib's SNMP implementation is selected, GnuGk starts a standalone SNMP agent. **NOTE**: Only one SNMP agent can bind to UDP/161, so you might have to use a non-standard port if you are using another SNMP agent on your server.

NOTE: By default, PTLib enables SNMP during the configuration process, so if SNMP doesn't work ensure that your PTLib `wasn't` compiled with `-disable-snmp`

Using PTLib rather than the fully-featured Net-SNMP means that only traps and GET requests are supported. See below for the additional switches which are required when using PTLib for SNMP.

The `Windows` implementation integrates as a sub-agent into Windows' SNMP service. This implementation is incomplete. You can use the `PTLib` implementation on Windows, for the sam elimited SNMP support as on Unix. Please contact the authors if you need full SNMP support on Windows.

12.10.1 GNU Gatekeeper Enterprise MIB

The GNU Gatekeeper Project was assigned the enterprise number 27938 by IANA (Internet Assigned Number Authority), so all of GnuGk's OIDs are under 1.3.6.1.4.1.27938.

The formal MIB specification (SMIv2) can be found in the file 'gnugk.mib' that is distributed with GnuGk. You might want to import it into your SNMP management software to see symbolic names for GnuGk's OIDs.

The following OIDs are available:

- 1.3.6.1.4.1.27938.11.1.1 GnuGk version (GET)
- 1.3.6.1.4.1.27938.11.1.2 GnuGk version with module flags (GET)
- 1.3.6.1.4.1.27938.11.1.3 Current number of endpoint registrations (GET)
- 1.3.6.1.4.1.27938.11.1.4 Number of ongoing calls (GET)
- 1.3.6.1.4.1.27938.11.1.5 Trace level (GET, SET)
- 1.3.6.1.4.1.27938.11.1.6 CatchAll destination (GET, SET)
- 1.3.6.1.4.1.27938.11.1.7 Total calls since startup (GET)
- 1.3.6.1.4.1.27938.11.1.8 Successful calls since startup (GET)

All of these OIDs are scalars, so please remember to append '.0' when querying them eg. with 'snmpget'.

Note: Setting the CatchAll destination via SNMP will update the config file to make the change permanent. Setting the trace level is a temporary setting.

The following traps are defined:

- 1 GnuGk started
- 2 GnuGk stopped

- 3 Config Reload
- 4 Module creation failed
- 5 Database operation failure
- 6 IO operation failure
- 7 General error
- 8 Authentication error
- 9 Message encoding/decoding error
- 10 Network error
- 11 Neighbor error

Traps may have 3 optional data elements:

- Severity level (OID 1.3.6.1.4.1.27938.11.2.1): Error=1, Warning=2, Info=3
- Group (OID 1.3.6.1.4.1.27938.11.2.2): General=1, Network=2, Database=3, Accounting=4, Authentication=5, Configuration=6
- Display string (OID 1.3.6.1.4.1.27938.11.2.3)

12.10.2 Configuring Net-SNMP

Net-SNMP configuration is usually found in /etc/snmp/snmpd.conf. This configuration file defines access control rules for the SNMP manager and where settings such as readonly and read-write community names are defined, where to send traps, etc.

Depending on which trap host definition you use, Net-SNMP will convert the traps to the appropriate version. Use 'trapsink' if you want to send version 1 traps, 'trap2sink' if you want to send version 2 traps and 'informsink' for SNMP v3 inform messages.

Please make sure AgentX support is enabled in the Net-SNMP daemon.

Simple example of a snmpd.conf for SNMP version 2c:

```
# server location and contact
syslocation Server Room
syscontact Sysadmin (root@example.com)

# read-only access only from this network, access to all MIBs
rocommunity public 192.168.1.0/24
# read-write access only from this network, restricted to the GnuGk MIB
rwcommunity mysecret 192.168.1.0/24     1.3.6.1.4.1.27938

# send traps as version 2 to this host with community string 'public'
trap2sink   192.168.1.64    public

# enable AgentX support
master agentx
agentxsocket tcp:localhost:705
```

For SNMP version 3 the config file could look like this:

```
# server location and contact
syslocation Server Room
syscontact Sysadmin (root@example.com)

# read-only access for user 'peter'
rouser peter
# full read-write access for user 'paul'
rwuser paul
# read-write access only for the GnuGk MIB for user 'mary'
rwuser mary auth 1.3.6.1.4.1.27938

# create the user accounts and set passwords
createUser peter MD5 peterpeter DES
createUser paul MD5 paulpaul DES
createUser mary MD5 marymary DES

# send traps as version 3 Inform messages with community string 'public'
informsink   192.168.1.64    public

# enable AgentX support
master agentx
agentxsocket tcp:localhost:705
```

You can also configure the Net-SNMP based agent to run standalone without the Net-SNMP daemon, but this is not suggested.

- `Standalone=1`
  Default: `0`
  Run SNMP-Agent standalone, and don't connect to AgentX master agent.

12.10.3 Using PTLib's SNMP implementation

PTlib supports SNMP version 1 and 2 GET requests and will always send version 1 traps in a version 2c message. GETNEXT (for 'walk') and SET requests are not supported.

When using PTLib, you can use these additional switches:

- `TrapHost=192.168.1.100`
  Default: `none`
  Define the trap destination host. No traps will be sent if no host is defined. You may specify an IP or DNS name.

- `TrapCommunity=public`
  Default: `public`
  Define the community string for traps.

- `AgentListenIP=192.168.1.100`
  Default: `127.0.0.1`
  Define the IP the SNMP agent should listen on.

- `AgentListenPort=11161`
  Default: `161`
  Define the UDP port the SNMP agent should listen on.

- `AllowRequestsFrom=192.168.1.0/22`
  Default: `n/a`
  A comma separated list of IPs or networks which are allowed to send us SNMP requests.

12.11 Section [TLS]

Using TLS (transport layer security), you can encrypt the Q.931 signalling channel (and eg. H.245 when it is tunneled). When TLS is enabled, GnuGk will listen on port 1300 for TLS connections.

Each leg of the call can use not not use TLS individually. For example one endpoint might encrypt its signalling to the GNU Gatekeeper and the 2nd half of the call going out to another endpoint may be unencrypted.

You have to define manually which endpoints use TLS when we call them, by setting a switch in their [EP::...] section. Similar you can enable TLS for neighbor and parent gatekeeper. Since not many endpoints support TLS encryption, so this feature can be very usefull to secure "trunk" connections or traversal zones to branch offices etc.

Using TLS is also important to avoid man-in-the-middle attacks on H.235.6 media encryption.

Note: Currently you must use H.245 tunneling when using TLS, consider H245TunnelingTranslation=1 to tunnel H.245 between gatekeepers when not all of your endpoints enable tunneling natively.

- `EnableTLS=1`
  Default: `0`
  Enable TLS support. GnuGk must have been compiled with OpenSSL support.
- `PrivateKey=server_key.pem`
  Default: `tls_private_key.pem`
  File with private key for this server.
- `Certificates=server_cert.pem`
  Default: `tls_certificate.pem`
  File with certificate for this server.
- `CAFile=root_cert.pem`
  Default: `n/a`
  File with root CA certificates.
- `CADir=/etc/certs/`
  Default: `n/a`
  Directory where to search for root CA certificates.
- `Passphrase=whatever`
  Default: `n/a`
  The passphrase to open the above certificate files, if you have set one.
- `CipherList=ALL:!ADH:!LOW:!EXP:!MD5:!RC4:!SHA1:!ECDH:!ECDSA:@STRENGTH`
  Default: `ALL:!ADH:!LOW:!EXP:!MD5:!RC4:!SHA1:!ECDH:!ECDSA:@STRENGTH`
  Set the OpenSSL cipher list to be used for TLS connections.
- `CheckCertificateIP=1`
  Default: `0`
  In addition to checking if a peer certificate is signed correctly, GnuGk can check if one of the X.509 extensions contains a DNS name or if the commonName in the certificate match the IP, the connections actually comming from.

Not all certificates contain proper DNS names and if your peers are behind a NAT you might not be able to see their true IPs. Also DNS lookups may be slow, so this check is disabled by default.

- `RequireRemoteCertificate=1`
  Default: `1`
  WARNING: USE SWITCH WITH CAUTION. It is highly recommended that TLS be established between 2 parties that both authenticate eachother via digital certificates. However there are use cases where generation and distribution of certificates to remote/distant endpoints becomes difficult or impossible. This switch when set to 0 will only authenticate TLS connections that the gatekeeper initiates and NOT ones it receives. For this reason it should ONLY be used in a gatekeeper environment where other security is present such as defining explicit Neighbors in Neighbors policy. For Endpoints it should ONLY be used with trusted endpoints and in conjunction with H.460.17 (recommended) and .18 where the endpoint always initiates the TLS connection and can vertify the certificate of the gatekeeper. If H.460.18 is used it is highly recommended to set [RoutedMode]NATStdMin=18 to ensure all endpoints MUST support H.460.18.

  This switch is only available if GnuGk is compiled with –eanble-weaktls and might be removed from future versions.

12.11.1 Generating keys and certificates with OpenSSL

You don't have to trust the same CAs (certification authorities) that you are using for web browsing etc. By default GnuGk ignores the root certificates installed with OpenSSL on your server, but you can enable them by setting CADir=.

To have tight control over who you trust, you can generate your own CA using OpenSSL and only allow connections with certificates signed by your CA. Set you own CA with the CAFile= switch and leave CADir= unset.

To follow the step-by-step instructions, you need a definition file for your CA, called root.cnf and one for the server called server.cnf (see below). While following these steps, you will be asked a few times for the passphrase for the generated files. You should use the same for all of them and add it to your GnbuGk config with the Passphrase= switch.

First, generate your CA and self-sign it:

```
openssl req -newkey rsa:2048 -sha256 -keyout root_key.pem -out root_req.pem -config root.cnf
openssl x509 -req -in root_req.pem -sha256 -extfile root.cnf -days 365 -extensions certificate_extensions
```

Then generate one key for each server or endpoint and sign them with the CA certificate:

```
openssl req -newkey rsa:2048 -sha256 -keyout server_key.pem -out server_req.pem -config server.cnf -reqext
openssl x509 -req -in server_req.pem -sha256 -extfile root.cnf -days 365 -extensions certificate_extension
```

And then repeat to generate as many certificates as necessary.

The certificates generated above are valid for 365 days. Make sure you replace them before they expire, otherwise you won't be able to make calls!

To check then content of a generated certificate (eg. to see if it has expired or if the DNS name is OK), you can use

```
openssl x509 -text -in server_cert.pem
```

Here is a sample root.cnf file that you can adapt:

```
[ ca ]
default_ca       = gnugk_ca

[ gnugk_ca ]
dir              = /opt/gnugk-ca
certificate      = $dir/cacert.pem
database         = $dir/index.txt
new_certs_dir    = $dir/certs
private_key      = $dir/private/cakey.pem
serial           = $dir/serial

default_crl_days = 7
default_days     = 365
default_md       = sha256

policy           = gnugk_ca_policy
x509_extensions  = certificate_extensions

[ gnugk_ca_policy ]
commonName             = supplied
stateOrProvinceName    = supplied
countryName            = supplied
emailAddress           = supplied
organizationName       = supplied
organizationalUnitName = optional

[ req ]
default_bits        = 2048
default_keyfile     = privkey.pem
default_md          = sha256

prompt              = no
distinguished_name  = req_distinguished_name
x509_extensions     = req_extensions

# the following sections are specific to the request we're building

[ certificate_extensions ]
basicConstraints = CA:true
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always

[ req_distinguished_name ]
countryName         = US
stateOrProvinceName = Virginia
localityName        = Fairfax
organizationName    = Zork.org
commonName          = GnuGk Root CA

[ req_extensions ]
basicConstraints = CA:true
```

Here is a sample server.cnf file that you can adapt:

```
[ ca ]
```

```
default_ca       = gnugk_ca

[ gnugk_ca ]
dir              = /opt/gnugk-ca
certificate      = $dir/cacert.pem
database         = $dir/index.txt
new_certs_dir    = $dir/certs
private_key      = $dir/private/cakey.pem
serial           = $dir/serial

default_crl_days = 7
default_days     = 365
default_md       = sha256

policy           = gnugk_ca_policy
x509_extensions  = certificate_extensions

[ gnugk_ca_policy ]
countryName            = supplied
stateOrProvinceName    = supplied
localityName           = supplied
organizationName       = supplied
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional

[ req ]
default_bits        = 2048
default_keyfile     = privkey.pem
default_md          = sha256

prompt              = no
distinguished_name  = req_distinguished_name
x509_extensions     = req_extensions

# the following sections are specific to the request we're building

[ certificate_extensions ]
basicConstraints = CA:false
subjectAltName = DNS:gk1.zork.org

[ req_distinguished_name ]
countryName            = US
stateOrProvinceName    = Virginia
localityName           = Fairfax
organizationName       = Zork.org
commonName             = gatekeeper.zork.org

[ req_extensions ]
basicConstraints = CA:true
subjectAltName = DNS:gk1.zork.org
```

13 Monitoring the Gatekeeper

13.1 Status Port

The status port is the external interface for monitoring and controlling the gatekeeper. The gatekeeper will send out messages about ongoing calls to all connected clients and it can receive commands via this interface.

Access to the status port is restricted by the rules in 4.6 (GkStatus::Auth). For security reasons, the default is not to allow any access until you have configured 4.6 (GkStatus::Auth).

The messages sent by the gatekeeper to the status port are grouped into three **output trace levels**: (These trace levels only apply to what is shown on the status port. Don't confuse them with the trace level for GnuGk's trace file.)

- Level 0

  Reload notifications and direct replies to entered commands.

- Level 1

  Reload notifications, direct replies to entered commands, CDRs and Route Requests.

- Level 2

  Output everything (reload notifications, direct replies to entered commands, CDRs, Route Requests, RAS, ...). This is the **default** output level.

The client connected to the status port can choose the output level it is interested in.

The interface is a simple TCP port (default: 7000) which you can connect to with telnet or another client. One example of a different client is the Java GUI, aka GkGUI. Another example is the Automatic Call Distribution application, aka GnuGk ACD.

13.1.1 Application Areas

What you do with the powers of the Status Interface is up to you, but here are a few ideas:

- Call Monitoring
- Monitoring the registered endpoints
- Graphical User Interface for GnuGk

  See GkGUI.

- Call Routing

  See GnuGk ACD.

- Billing Applications

  Analyze the CDR messages and forward them to a billing application.

- Interfacing external extensions

  If you don't want to publish the source code to additional features, just publish the core functionality and interface to it through the status interface and keep the external part private.

13.1.2 Examples

Suppose you are just interested in the CDRs (call detail records) and want to process them as a batch at regular intervals.

Here is a simple Perl script (`gnugk_cdr.pl`) that starts the gatekeeper and also forks a very simple client for the Status Interface and writes just the CDRs into a logfile. You’ll have to modify it a little to fit your needs.

```
#!/usr/bin/perl
# sample program that demonstrates how to write the CDRs to a log file
use strict;
use IO::Socket;
use IO::Handle;

my $logfile = "/home/jan/cdr.log";      # CHANGE THIS
my $gk_host = "localhost";
my $gk_port = 7000;
my $gk_pid;

if ($gk_pid = fork()) {
        # parent will listen to gatekeeper status
        sleep(1);       # wait for gk to start
        my $sock = IO::Socket::INET->new(PeerAddr => $gk_host, PeerPort => $gk_port, Proto => ’tcp’)
        if (!defined $sock) {
                die "Can’t connect to gatekeeper at $gk_host:$gk_port";
        }
        $SIG{HUP} = sub { kill 1, $gk_pid; };   # pass HUP to gatekeeper
        $SIG{INT} = sub { close (CDRFILE); kill 2, $gk_pid; };  # close file when terminated

        open (CDRFILE, ">>$logfile");
        CDRFILE->autoflush(1);  # don’t buffer output
        while (!$sock->eof()) {
                my $msg = $sock->getline();
                $msg = (split(/;/, $msg))[0];   # remove junk at end of line
                my $msgtype = (split(/\|/, $msg))[0];
                if ($msgtype eq "CDR") {
                        print CDRFILE "$msg\n";
                }
        }
        close (CDRFILE);
} else {
        # child starts gatekeeper
        exec("gnugk");
}
```

Keep in mind that this is just an example to show the usage of the status port. You can use the FileAcct module to log CDRs in a production system.

13.1.3 Java GUI for the Gatekeeper

Developed by Jan Willamowius.

You can monitor the registrations and calls that go through the gatekeeper. A right-click on a button gives you a pop up menu for that endpoint.

This GUI works with Java 1.0 built into most web browsers. For security reasons the GUI must be run as a standalone application or served by a web server on the same IP number as the gatekeeper (you cannot run it as an applet via a local file).

The program is available at *GnuGk.org* `<http://www.gnugk.org/h323gui.html>`

13.2 Commands (Reference)

This section lists all commands that you can issue to the status port (manually or with an external application). Commands are not case-insensitive, but parameters may be.

Entering `help` or `h` will display a list of all available commands.

- `Reload`
  Reload the configuration.

  Reloading the configuration will not terminate existing calls, and any change to settings will only take effect on new calls.

  You can add an optional parameter to reload only a part of your configuration:

  - AcctConfig - reload only the accounting config
  - AuthConfig - reload only the authentication config
  - CapConfig - reload only the CapacityControl rules
  - EpConfig - reload only the endpoint config (permanent endpoints, endpoint section, call table settings)

  Example:

  ```
  Reload
  Full Config reloaded.

  Reload EpConfig
  EP Config reloaded.
  ```

- `Shutdown`
  Terminate the gatekeeper. Can be disabled by Shutdown=forbid in section 4.6 ([GkStatus::Auth]).

- `Version`, `v`
  Show the version and OS information of the gatekeeper.

- `Statistics`, `s`
  Show the statistics information of the gatekeeper.

  Example:

  ```
  Statistics
  -- Endpoint Statistics --
  Total Endpoints: 307  Terminals: 278  Gateways: 29  NATed: 0
  Cached Endpoints: 0  Terminals: 0  Gateways: 0
  -- Call Statistics --
  ```

```
Current Calls: 7 Active: 7 From Neighbor: 4 From Parent: 0 Proxied: 3
Total Calls: 1151  Successful: 485  From Neighbor: 836  From Parent: 0  Proxied: 193  Peak:  17 at
Startup: Tue, 26 Nov 2013 18:45:35 +04:00   Running: 0 days 02:34:15
;
```

- `ResetCallCounters`
  Reset the statistics counters for total calls, successful calls, neighbor calls and parent calls to zero.

- `PrintAllRegistrations`, `r`, `?`
  Show all registered endpoints.

  Format:

```
AllRegistrations
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
...
Number of Endpoints: n
;
```

  Example:

```
AllRegistrations
RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
RCF|10.0.1.43:1720|613:dialedDigits=Jacky Tsai:h323_ID|terminal|1328_endp
RCF|10.0.1.55:1720|705:dialedDigits=Sherry Liu:h323_ID|terminal|1333_endp
Number of Endpoints: 3
;
```

- `PrintAllRegistrationsVerbose`, `rv`, `??`
  Show details of all registered endpoints.

  Format:

```
AllRegistrations
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
Registration_Time C(Active_Call/Connected_Call/Total_Call) <r> (NAT type) bw:Bandwidth/Max_Bandwid
[Prefixes: ##] (gateway only)
...
Number of Endpoints: n
;
```

  Example:

```
AllRegistrations
RCF|10.0.1.8:1720|Accel-GW2:h323_ID|gateway|1322_endp
Wed, 26 Jun 2002 16:40:03 +0800 C(1/5/33) <1> bw:0/10240
Prefixes: 09,002
RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
Wed, 26 Jun 2002 16:40:55 +0800 C(0/32/39) <1> (H.460.18) bw:7680/10240
RCF|10.0.1.66:1720|716:dialedDigits=Vicky:h323_ID|terminal|1425_endp
Wed, 26 Jun 2002 16:40:58 +0800 C(1/47/53) <1> (H.460.17) bw:0/10240
Number of Endpoints: 3
;
```

- `PrintAllCached`, `rc`
  Print list of all cached out-of-zone endpoints.

- `PrintCurrentCalls`, `c`, `!`
  Show all current calls using the same ACF syntax as in call establishment. Also shows how media is being routed.

Format:

```
CurrentCalls
Call No. # | CallID | Call_Duration | Left_Time
Dialed_Number
ACF|Caller_IP:Port|Caller_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered|MediaRoute;
ACF|Callee_IP:Port|Callee_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered|MediaRoute;
...
Number of Calls: Current_Calls Active: Active_Calls From Neighbor: Calls_From_Neighbor \
From Parent: Calls_From_Parent Proxied: Proxied_Calls
;
```

Example:

```
CurrentCalls
Call No. 29 | CallID bd c6 17 ff aa ea 18 10 85 95 44 45 53 54 77 77 | 109 | 491
Dial 0953378875:dialedDigits
ACF|10.0.1.49:1720|4048_CGK1|25263|frank:h323_ID|gunter:h323_ID|false|Proxy;
ACF|10.1.1.1:1720|4037_CGK1|25263|gunter:h323_ID|frank:h323_ID|true|Proxy;
Call No. 30 | CallID 70 0e dd c0 9a cf 11 5e 00 01 00 05 5d f9 28 4d | 37 | 563
Dial 0938736860:dialedDigits
ACF|10.0.1.48:1032|4041_CGK1|11896|sue:h323_ID|peter:h323_ID|false|-;
ACF|10.1.1.1:1720|4037_CGK1|11896|peter:h323_ID|sue:h323_ID|true|-;
Number of Calls: 2 Active: 2 From Neighbor: 0 From Parent: 0 Proxied: 1
;
```

- `PrintCurrentCallsVerbose`, `cv`, `!!`
  Show details of all current calls.

Format:

```
CurrentCalls
Call No. # | CallID | Call_Duration | Left_Time
Dialed_Number
ACF|Caller_IP:Port|Caller_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered|MediaRoute;
ACF|Callee_IP:Port|Callee_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered|MediaRoute;
# Caller_Aliases|Callee_Aliases|Bandwidth|Connected_Time <r> bw:Bandwidth
...
Number of Calls: Current_Calls Active: Active_Calls From Neighbor: Calls_From_Neighbor \
From Parent: Calls_From_Parent Proxied: Proxied_Calls
;
```

Example:

```
CurrentCalls
Call No. 48 | CallID 7d 5a f1 0a ad ea 18 10 89 16 00 50 fc 3f 0c f5 | 30 | 570
Dial 0225067272:dialedDigits
ACF|10.0.1.200:1720|1448_endp|19618|frank:h323_ID|gunter:h323_ID|false|Proxy;
ACF|10.0.1.7:1720|1325_endp|19618|gunter:h323_ID|frank:h323_ID|true|Proxy;
# Sherry:h323_ID|Accel-GW1:h323_ID|200000|Wed, 26 Jun 2002 17:29:55 +0800 <2> bw:3840
Number of Calls: 1 Active: 1 From Neighbor: 0 From Parent: 0 Proxied: 1
;
```

- `PrintCurrentCallsPorts`
  Show the dynamically allocated ports for each ongoing call that are used for **incoming** packets.

Format:

```
CurrentCallsPorts
Call No. # | CallID | Call_Duration | Dialed_Number
```

```
Caller_IP:Port|SrcInfo|Callee_IP:Port|DestinationInfo
  PortType IP:port
;
```

Example:

```
CurrentCallsPorts
Call No. 1 | CallID b4 ef 4a e3 2a f8 e0 11 9f c6 00 1e c9 7e 69 ec | 62 | Dial peter:h323_ID
10.0.1.200:1720|frank:h323_ID|10.0.1.7:1720|gunter:h323_ID
  RTP 0.0.0.0:1024
  RTP 0.0.0.0:1025
  H.245 0.0.0.0:55674
;
```

- `PrintPrefixCapacities`, `printpc`
  Print the prefix capacities and current counter values for all endpoints or the specified alias.

  Format:

```
PrintPrefixCapacities [Alias]
PrefixCapacities
-- Endpoint: Alias (1.2.3.4:1720) --
Total calls = 0
prefix/capacity/curr: 125/5/0
-- Endpoint: Alias2 (1.2.3.5:1720) --
Total calls = 0
prefix/capacity/curr: 125/5/0
;
```

  Example:

```
PrintPrefixCapacities OpenMCU
PrefixCapacities
-- Endpoint: OpenMCU (192.168.1.100:1720) --
Total calls = 0
prefix/capacity/curr: ^(123|124)/2/0
prefix/capacity/curr: 125/5/0
;
```

- `printcc`
  Print the current counters for all CapacityControl rules.

- `Find`, `f`
  Find a registered endpoint by an alias or a prefix. To find an alias of the specified type (h323_ID, dialedDigits), prepend the alias type name (h323, e164, url, email) to the alias, followed by a colon.

  Format:

```
Find Alias
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
;
```

  Example:

```
f 800
RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
;
f 801
Alias 801 not found!
f h323:Wei
RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
;
```

- `FindVerbose`, `fv`
  Find details of a registered endpoint by an alias or a prefix. To find an alias of the specified type (h323_ID, dialedDigits), prepend the alias type name (h323, e164, url, email) to the alias, followed by a colon.

  Format:
  ```
  FindVerbose Alias
  RCF|IP:Port|Aliases|Terminal_Type|EndpointID
  Registration_Time C(Active_Call/Connected_Call/Total_Call) <r>
  [Prefixes: ##] (gateway only)
  ;
  ```
  Example:
  ```
  fv 02
  RCF|10.0.1.100:1720|TFN:h323_ID|gateway|4037_CGK1
  Wed, 26 Jun 2002 17:47:29 +0800 C(0/84/120) <1>
  Prefixes: 02,09
  ;
  ```

- `UnregisterIP`
  Forcefully unregister an endpoint by IP and call signaling port. If you don't specify a call signal port, GnuGk will unregister the first endpoint it finds on the IP number

  Format:
  ```
  UnregisterIP IP[:Port]
  ```
  Example:
  ```
  UnregisterIP 10.0.1.31:1720
  URQ|10.0.1.31:1032|1326_endp|maintenance;
  Endpoint 10.0.1.31:1720 unregistered!
  ```

- `UnregisterAlias`
  Forcefully unregister an endpoint by one of its aliases. To match an alias of the specified type (h323_ID, dialedDigits), prepend the alias type name (h323, e164, url, email) to the alias, followed by a colon.

  Format:
  ```
  UnregisterAlias Alias
  ```
  Example:
  ```
  UnregisterAlias 601
  URQ|10.0.1.31:1032|1326_endp|maintenance;
  Endpoint 601 unregistered!
  ```

- `UnregisterAllEndpoints`
  Forcefully unregister all registered endpoints.

  Format:

  Example:
  ```
  UnregisterAllEndpoints
  URQ|10.0.1.7:1024|1325_endp|maintenance;
  URQ|10.0.1.8:1024|1322_endp|maintenance;
  URQ|10.0.1.32:1032|1324_endp|maintenance;
  URQ|10.0.1.36:1032|1323_endp|maintenance;
  URQ|10.0.1.42:1032|1318_endp|maintenance;
  Done
  ;
  ```

- `DisconnectCall`
  Disconnect a call with given number (internal, gatekeeper assigned call number, not the caller's, callee's phone number).

  Format:

  ```
  DisconnectCall Number
  ```

  Example:

  ```
  DisconnectCall 1533
  ```

- `DisconnectCallId`
  Disconnect a call with given call ID.

  Format:

  ```
  DisconnectCallId CallId
  ```

  Example:

  ```
  DisconnectCallId ee-ab-8f-81-58-57-df-11-95-39-00-1e-c9-7e-69-ec
  ```

- `DisconnectIP`
  Disconnect all calls of an endpoint by IP and call signaling port. If you don't specify a call signal port, GnuGk will disconnect the first endpoint it finds on the IP number

  Format:

  ```
  DisconnectIP IP[:Port]
  ```

  Example:

  ```
  DisconnectIP 10.0.1.31:1720
  ```

- `DisconnectAlias`
  Disconnect all calls of a registered endpoint by one of its aliases. To match an alias of the specified type (h323_ID, dialedDigits), prepend the alias type name (h323, e164, url, email) to the alias, followed by a colon.

  Format:

  ```
  DisconnectAlias Alias
  ```

  Example:

  ```
  DisconnectAlias 601
  ```

- `DisconnectEndpoint`
  Disconnect all calls of a registered endpoint by one of its endpoint identifier.

  Format:

  ```
  DisconnectEndpoint ID
  ```

  Example:

  ```
  DisconnectEndpoint 5624_endp
  ```

- `ClearCalls`
  Disconnect all calls on the gatekeeper.

- `GK`
  Show the information of the parent gatekeeper.

- `Trace`
  Set the status interface output trace level. It controls which messages are sent to this client:

  - `trace 0` or `trace min`
    Only direct responses to commands and reload notifications.
  - `trace 1`
    CDRs, direct responses to commands and reload notifications.
  - `trace 2` or `trace max`
    Show all (RAS, CDRs, direct responses to commands, reload notifications, etc).

- `Debug`
  Only used for debug purpose. Options:
  - `trc [+|-|n]`
    Show/modify trace level.
  - `cfg`
    Read and print a list of all sections.
  - `cfg all`
    Read and print the content of all config sections.
  - `cfg SEC`
    Read and print a config section.
  - `cfg SEC PAR`
    Read and print a config parameter in a section.
  - `set SEC PAR VAL`
    Write a config value parameter in a section. (Beware that you can't set values containing spaces using this command!)
  - `remove SEC PAR`
    Remove a config value parameter in a section.
  - `remove SEC`
    Remove a section.
  - `printrm VERBOSE`
    Print all removed endpoint records.

  Example:

  ```
  debug trc 3
  debug set RoutedMode H245Routed 1
  ```

- `SetLog`
  Send trace output to another file.

  Format:

  ```
  Setlog [filename]
  ```

  Example:

  ```
  Setlog /tmp/trace.log
  ```

- `RotateLog`
  Rotate the log file.
- `Who`
  Show all people on the status port. First field is the session id, which can be used to disconnect a user through the DisconnectSession command.
- `DisconnectSession`
  Disconnect a user from the status port.

Format:

```
DisconnectSession [session id]
```

Example:

```
DisconnectSession 2
```

- `Yell`, `y`
  Send a message to all status clients.

  Format:

  ```
  Yell [message text]
  ```

  Example:

  ```
  Yell Config reload in 5 minutes.
  ```

- `RouteReject`
  Terminate this call on a virtual queue. This command is used as a response to a RouteRequest event (see below). CallingEndpointID and CallRef must be passed back as they are in the corresponding RouteRequest. The CallID parameter is optional; if it is given it has to be the same format as signaled by RouteRequest.

  Format:

  ```
  RouteReject CallingEndpointID CallRef [CallID]
  ```

  Example:

  ```
  RouteReject endp_4711 1234
  ```

- `RouteToAlias`, `rta`
  Route this call on a virtual queue to the specified alias. This command is used as a response to a RouteRequest event (see below). CallingEndpointID and CallRef must be passed back as they are in the corresponding RouteRequest. The CallID parameter is optional; if it is given it has to be the same format as signaled by RouteRequest. As additional parameter you can set the CLI for the caling party.

  Format:

  ```
  RouteToAlias Alias CallingEndpointID CallRef [CallID [CLI]]
  ```

  Example:

  ```
  RouteToAlias Suzi endp_4711 1234
  ```

- `RouteToGateway`, `rtg`
  Route this call on a virtual queue to the specified alias and set the destinationSignalAddress. This command is used as a response to a RouteRequest event (see below). You can use this command to route calls to out-of-zone gateways or MCUs not registered with the gatekeeper. Make sure that the 'vqueue' and 'explicit' policy is in effect for these calls. CallingEndpointID and CallRef must be passed back as they are in the corresponding RouteRequest. The CallID parameter is optional; if it is given it must be the same format as signaled by RouteRequest. As additional parameter you can set the CLI for the caling party. The alias parameter is required, but if you use a dash ("-") as alias name, no alias will be set in the destination, just the IP.

  Format:

  ```
  RouteToGateway Alias IP:Port CallingEndpointID CallRef [CallID [CLI]]
  ```

  Example:

  ```
  RouteToGateway Suzi 192.168.0.50 endp_4711 1234
  ```

- `RouteToInternalGateway`
  Same as RouteToGateway, but does not tell caller about about an updated destination alias.

- `BindAndRouteToGateway`
  This command is similar to RouteToGateway, but you can also specify which IP of a multi-homed server to use for the outgoing call.

  Format:

  ```
  BindAndRouteToGateway IP Alias IP:Port CallingEndpointID CallRef [CallID [CLI]]
  ```

  Example:

  ```
  BindAndRouteToGateway 192.168.0.2 Suzi 192.168.0.50 endp_4711 1234
  ```

- `BindAndRouteToGateway`
  Same as BindAndRouteToGateway, but does not tell caller about about an updated destination alias.

- `SendProceeding`
  Send a CallProceeding message to the caller. The only time this makes sense is after a RouteRequest event for an unregistered call. Otherwise a status port application won't know if a Setup message has been sent but that the call is not yet established.

  Format:

  ```
  SendProceeding CallID
  ```

  Example:

  ```
  SendProceeding 40-06-dd-98-22-37-52-40-8c-b0-92-0e-18-60-99-07
  ```

- `Exit`, `Quit`, `q`, `Ctrl-D`
  Quit the status port.

- `TransferCall`
  Transfer an established call from endpoint A to endpoint B.

  The call that shall be transfered is selected by the call ID and the string "caller" or "called" is used to specify the which end of the call shall be transferred.

  You can choose the transfer method with the optional last parameter. The call transfer works only with endpoints that properly support Q.931 Facility callForwarded or routeCallToMC messages (so it doesn't work with Netmeeting).

  Format:

  ```
  TransferCall <call-id> <CALLER | CALLED> <destination> [<FacilityForward | FacilityRouteCallToMC>
  ```

  Example:

  ```
  TransferCall ee-ab-8f-81-58-57-df-11-95-39-00-1e-c9-7e-69-ec caller Peter
  ```

- `RerouteCall`
  EXPERIMENTAL: Gatekeeper based call transfer using TCS0 pause and reroute.

  Format:

  ```
  RerouteCall <call-id> <CALLER|CALLED> <destination>
  ```

  Example:

  ```
  RerouteCall 40-06-dd-98-22-37-52-40-8c-b0-92-0e-18-60-99-07 CALLER 192.168.1.222
  ```

- `MakeCall`
  Generate a new call from source to destination alias. You can also specify an IP number as destination. This is done by establishing a call from a pseudo endpoint in the gatekeeper to the source alias/number and then transferring the call from the gatekeeper endpoint to the destination. The method how this call transfer is done can be configured.

  See 12.7 ([CTI::MakeCall]) for configuration options.

  Format:

  ```
  MakeCall Source-Alias Destination-Alias
  ```

  Example:

  ```
  MakeCall 1234 5678
  MakeCall joe 192.168.6.1
  ```

- `GetAuthInfo,gai`
  Gather information from a specific authentication module (if it provides such information) and displays it on the status port.

  Format:

  ```
  GetAuthInfo ModuleName
  ```

  Example:

  ```
  GetAuthInfo RadAliasAuth
  ```

- `GetAcctInfo,gci`
  Gather information from a specific accounting module (if it provides such information) and displays it on the status port.

  Format:

  ```
  GetAcctInfo ModuleName
  ```

  Example:

  ```
  GetAcctInfo SqlAcct
  ```

- `PrintEndpointQoS`
  Display QoS values for all endpoints. The values are collected through H.460.9 and/or proxied RTCP messages, so at least one of these features must be enabled to have packet loss or jitter values. If an endpoint has more than one active call, the packet loss and jitter values are from one randomly selected call.

  Format:

  ```
  QoS|<endpoint aliases>|<last contact from endpoint>|<num calls>|<audio Rx packet loss percent>|<a
  ```

  Example:

  ```
  EndpointQoS
  QoS|8001:dialedDigits=Peter:h323_ID|2011-02-10 T 09:23:08 Z|1|0.00%|0|0.00%|0
  QoS|Mary:h323_ID|2011-02-10 T 09:23:08 Z|1|0.00%|0|0.00%|0
  Number of Endpoints: 2
  ;
  ```

- `PrintEventBacklog`
  Print the saved status port events in the event backlog. To configure the event backlog see 4.5 ([Gatekeeper::Main] StatusEventBacklog).

13.3 Messages (Reference)

The section describes the messages output to the status interface.

- `GCF|IP|Aliases|Endpoint_Type;`
  The gatekeeper receives a GatekeeperRequest (GRQ) and responds with a GatekeeperConfirm (GCF).
- `GRJ|IP|Aliases|Endpoint_Type|RejectReason;`
  The gatekeeper receives a GatekeeperRequest (GRQ) and responds with a GatekeeperReject (GRJ).
- `RCF|IP:Port|Aliases|Endpoint_Type|EndpointID;`
  The gatekeeper receives a RegistrationRequest (RRQ) and responds with a RegistrationConfirm (RCF).
- `RRJ|IP|Aliases|Endpoint_Type|RejectReason;`
  The gatekeeper receives a RegistrationRequest (RRQ) and responds with a RegistrationReject (RRJ).
- `ACF|Caller_IP:Port|Caller_EndpointID|CRV|DestinationInfo|SrcInfo|IsAnswered|CallID|MediaRoute;`
  The gatekeeper receives an AdmissionRequest (ARQ) and responds with an AdmissionConfirm (ACF).
- `ARJ|Caller_IP:Port|DestinationInfo|SrcInfo|IsAnswered|RejectReason|CallID;`
  The gatekeeper receives an AdmissionRequest (ARQ) and responds with an AdmissionReject (ARJ).
- `DCF|IP|EndpointID|CRV|DisengageReason|CallID;`
  The gatekeeper receives a DisengageRequest (DRQ) and responds with a DisengageConfirm (DCF).
- `DRJ|IP|EndpointID|CRV|RejectReason|CallID;`
  The gatekeeper receives a DisengageRequest (DRQ) and responds with a DisengageReject (DRJ).
- `LCF|IP|EndpointID|DestinationInfo|SrcInfo;`
  The gatekeeper receives a LocationRequest (LRQ) and responds with a LocationConfirm (LCF).
- `LRJ|IP|DestinationInfo|SrcInfo|RejectReason;`
  The gatekeeper receives a LocationRequest (LRQ) and responds with a LocationReject (LRJ).
- `BCF|IP|EndpointID|Bandwidth;`
  The gatekeeper receives a BandwidthRequest (BRQ) and responds with a BandwidthConfirm (BCF).
- `BRJ|IP|EndpointID|Bandwidth|RejectReason;`
  The gatekeeper receives a BandwidthRequest (BRQ) and responds with a BandwidthReject (BRJ).
- `UCF|IP|EndpointID;`
  The gatekeeper receives an UnregistrationRequest (URQ) and responds with an UnregistrationConfirm (UCF).
- `URJ|IP|EndpointID|RejectReason;`
  The gatekeeper receives an UnregistrationRequest (URQ) and responds with an UnregistrationReject (URJ).
- `IRQ|IP:Port|EndpointID;`
  The gatekeeper sends an InfoRequest (IRQ) to an endpoint to query if it is still alive. The endpoint must immediately respond with an InfoRequestResponse (IRR).
- `URQ|IP:Port|EndpointID|Reason;`
  The gatekeeper sends an UnregistrationRequest (URQ) to an endpoint to cancel its registration. The endpoint shall respond with an UnregistrationConfirm (UCF).

- `CDR|CallNo|CallId|Duration|Starttime|Endtime|CallerIP|CallerEndId| \`
  `CalledIP|CalledEndId|DestinationInfo|SrcInfo|GatekeeperID;`
  After a call disconnected, the call detail record is shown (in one line).

- `RouteRequest|CallerIP:Port|CallerEndpointId|CallRef|VirtualQueue|CallerAlias|CallID|CalledIP:Port|V`
  Request for an external application to route an incoming call on a virtual queue. This can be done with a RouteToAlias/RouteToGateway or RouteReject command.

13.4 Status Port Filtering

Status port filtering facilitates control of the amount and type of output messages shown to the end user. Filtering is done using regular expressions which are used to decide whether to include (show) or exclude (ignore) an output message. Filtering control is performed using the following set of commands:

- `addincludefilter REGEX`
  Adds regular expression to the include list

- `addexcludefilter REGEX`
  Adds regular expression to the exclude list

- `removeincludefilter INDEX`
  Removes filter at given INDEX from the include list

- `removeexcludefilter INDEX`
  Removes filter at given INDEX from the exclude list

- `filter 1|0`
  Enable/Disable message filtering

- `printincludefilters`
  Print include filter list

- `printexcludefilters`
  Print exclude filter list

In order to enable usage of predefined filters, a new section named 4.7 ([GkStatus::Filtering]) has been introduced. You may specify predefined filters to be loaded when the status port starts.

Example:

```
[GkStatus::Filtering]
IncludeFilter=.+
ExcludeFilter=.RQ
Enable=1
```

When filtering is enabled using the the `filter 1` command, all messages will be shown other than lines with ARQ, LRQ etc. You may also type the following into the status port:

```
addincludefilter .+
addexcludefilter .RQ
filter 1
```

Note that if you enable filtering when there are no include filters defined this will automatically exclude all message output!

Example to hide Tandberg's neighbor check and traversal zone keepalive messages:

```
[GkStatus::Filtering]
Enable=1
IncludeFilter=.+
ExcludeFilter=gatekeeper-monitoring-check
ExcludeFilter=SCR
```

There is an additional switch to only print RCF events for previously unregistered endpoints and supres

```
[GkStatus::Filtering]
NewRCFOnly=1
```

13.5 Status Port Message Format

The format of status port event messages may be altered to reorder or include options not included in the standard output format. NOTE: This section has no effect on the format of the response of status port commans, like eg. PrintAllRegistrationsVerbose.

The settings in this section may be updated by reloading the configuration while the gatekeeper is running.

- `Compact=1`
  Default: `0`
  Whether to use standard format or compact format parameters (mainly used with NATType variable)
- `RCF=%{IP:Port}|%{Aliases}|%{Endpoint_Type}|%{EndpointID}|%{NATType}|%{Vendor}`
  Default: `N/A`
  The RFC event has the following parameters available:
  - IP:Port - IP and port the registration was detected on (external NAT address if behind NAT)
  - Aliases - Complete list of aliases for the registration
  - Endpoint_Type - Endpoint type (ie terminal,gateway) of the registered endpoint
  - EndpointID - endpoint id assigned to the registration
  - NATType - NAT Method of registration (ie Native,GnuGk,H.460.17,H.460.18 etc)
  - Vendor - Vendor information of the registering device
- `URQ=%{IP:Port}|%{Aliases}|%{Endpoint_Type}|%{EndpointID}|%{NATType}|%{Vendor}|%{EndpointRASAddr}|%{U`
  Default: `N/A`
  The URQ event has the following parameters available:
  - IP:Port - IP and port the registration was detected on (external NAT address if behind NAT)
  - Aliases - Complete list of aliases for the registration
  - Endpoint_Type - Endpoint type (ie terminal,gateway) of the registered endpoint
  - EndpointID - endpoint id assigned to the registration
  - NATType - NAT Method of registration (ie Native,GnuGk,H.460.17,H.460.18 etc)
  - Vendor - Vendor information of the registering device
  - EndpointRASAddr - Endpoint RAS address
  - URQReason - Unregistration reason

Example:

```
[GkStatus::Message]
Compact=0
RCF=%{IP:Port}|%{Aliases}|%{Endpoint_Type}|%{EndpointID}|%{NATType}|%{Vendor}
URQ=%{IP:Port}|%{Aliases}|%{Endpoint_Type}|%{EndpointID}|%{NATType}|%{Vendor}|%{EndpointRASAddr}|%{URQ
```

14 Advanced Topics

This portion of the manual will cover advanced topics, such as compiling and debugging the GNU Gatekeeper.

14.1 Compiling GnuGk from CVS

The following instructions are an example of how to compile GnuGk from source on an Ubuntu platform.

First make sure your system is up-to-date and install the tools needed for the compile

```
sudo apt-get update
sudo apt-get install flex bison build-essential subversion cvs pkg-config automake linuxdoc-tools
```

Also make sure the "...-devel" packages for all databases or LDAP servers you want to use are installed.

NOTE: As of 2013-12-30, it's recommended that you do not use PTLib SVN; it is undergoing many changes that are incompatible with GnuGk. To compile the recommended PTLib 2.10.9, you need bison 2.x (bison 3.x won't work).

Get and compile PTLib from SourceForge:

```
cd ~
svn co http://svn.code.sf.net/p/opalvoip/code/ptlib/tags/v2_10_9 ptlib-v2.10.9/
cd ptlib-v2.10.9
export PTLIBDIR=~/ptlib-v2.10.9
./configure --enable-ipv6 --disable-odbc
wget -O src/ptlib/common/getdate.tab.c http://www.gnugk.org/download/getdate.tab.c # in case you hav
make optnoshared
```

Get and compile H323Plus:

```
cd ~
cvs -d:pserver:anonymous@h323plus.cvs.sourceforge.net:/cvsroot/h323plus login

(just press enter when prompted for password)

cvs -z3 -d:pserver:anonymous@h323plus.cvs.sourceforge.net:/cvsroot/h323plus co -P h323plus

cd h323plus
export OPENH323DIR=~/h323plus
./configure
make optnoshared
```

Get and compile GnuGk:

```
cd ~
cvs -d:pserver:anonymous@openh323gk.cvs.sourceforge.net:/cvsroot/openh323gk login

(just press enter when prompted for password)

cvs -z3 -d:pserver:anonymous@openh323gk.cvs.sourceforge.net:/cvsroot/openh323gk co -P openh323gk
```

```
cd openh323gk
./configure --enable-h46018
make optnoshared
```

Once the compile is finished, the binary can be found in the obj_linux_x86_s subdirectory.

At this time, because all libraries and GnuGk are running CVS and SVN versions of the software, in order to stay up-to-date, run the following:

```
cd ~/ptlib
svn update
cd ~/h323plus
cvs update
cd ~/openh323gk
cvs update
```

If any of the source files are changed, you have to recompile.

14.2 Tracing GnuGk

If GnuGk doesn't handle calls like you expect, you can enable tracing to see what GnuGk does internally. This should not be confused with connection to the status port and looking at the events ("telnet 127.0.0.1 7000"). Creating a trace file will reveal a lot more of the internal workings.

On the command line, start GnuGk with -ttttt and -o to write the trace to a file:

```
gnugk -c gnugk.ini -ttttt -o trace.log
```

If you have a lot of calls, trace.log can grow quite large, so make sure you disable it after you are done with testing, or at least reduce the trace level to 2 or 3 for production.

You can also enable tracing in your config file:

```
[Gatekeeper::Main]
TraceLevel=5

[LogFile]
Filename=trace.log
```

Or you can enable tracing through the status port:

```
setlog trace.log
debug trc 5
```

Doing it through the status port has the advantage that you won't interrupt ongoing calls and you can quickly turn it on or off.

The trace file will contain information detailing everything GnuGk does. To reduce it to a single call, you can search for the callID or write a small Perl script to extract only those messages you are interested in.

14.3 Debugging GnuGk (on Linux)

In order to use gdb with GnuGk, the software and libraries must be compiled with debug support.

You may follow the instructions above in obtaining the software, but the compile in each subdirectory must be:

```
make debugnoshared
```

Allow unlimited core dumps:

```
ulimit -c unlimited
```

Run GnuGk:

```
~/openh323/obj_linux_x86_64_d_s/gnugk -c your.ini
# wait for crash
gdb obj_linux_x86_64_d_s/gnugk core
bt
```

Once you've obtained a backtrace, post it to the mailing list.

Note: On some systems, the core dump is named "core.xxx" where xxx is the process number of the program that crashed.