OpenH323 Gatekeeper - The GNU Gatekeeper

Maintainer of this manual: *Jan Willamowius* <*jan@willamowius.de*>

This is the User Manual how to compile, install, configure and monitor OpenH323 Gatekeeper - The GNU Gatekeeper.

Contents

1 Introduction

1.1 About

OpenH323 Gatekeeper - The GNU Gatekeeper `<http://www.gnugk.org/>` is an open-source project that implements an H.323 gatekeeper. A gatekeeper provides call control services to the H.323 endpoints. It is an integral part of most useful internet telephony installations that are based on the H.323 standard.

According to Recommendation H.323, 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 most of these functions based on the *OpenH323* <http://sourceforge.net/projects/openh323> protocol stack.

Recommendation 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/>.

For a detailed description of what a gatekeeper does, see *here* <http://www.iec.org/online/tutorials/h323/topic06.html>.

1.2 Copyright

It is covered by the *GNU General Public License* (GNU GPL). In addition to that, we explicitly grant the right to link this code to the OpenH323 and OpenSSL library.

Generally speaking, the GNU GPL allows you to copy, distribute, resell or modify the software, but it requires that all derived works must be published under GNU GPL also. That means that you must publish full source for all extensions to the gatekeeper and for all programs you include the gatekeeper into. 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 via TCP with it. That way you only have to integrate the basic functionality into the gatekeeper (and provide source for that) and can keep other parts of your application private.

1.3 Name

The formal name of this project is *OpenH323 Gatekeeper - The GNU Gatekeeper*, short *GnuGk*. Please don't confuse it with other gatekeeper projects.

There are several open-source gatekeeper projects based on the OpenH323 protocol stack.

- *OpenGatekeeper* <http://opengatekeeper.sourceforge.net/>

  A gatekeeper available under MPL. The project has been inactive for a period of time now.

- *OpenGK* <http://sourceforge.net/projects/openh323>

  Only in a very primary grades.

- *OpenH323 Gatekeeper - The GNU Gatekeeper* <http://www.gnugk.org/>

  This one, also called GnuGk.

To have different gatekeepers with very similar names is really confusing for most users. Since our "OpenH323 Gatekeeper" was the first on the scene, it is not our fault that others have chosen similar names. But to make the distinction a little more clear without confusing people even more, we have decided to give the project a subtitle "OpenH323 Gatekeeper - The GNU Gatekeeper" and start using `gnugk` as name for executables.

1.4 Features

The version 2.2.4 contains the following new features and bugfixes:

- New call failover feature to retry failed calls on a different route.
- New Calling/CalledTypeOfNumber configuration variables in `RoutedMode` section and endpoint configuration.
- `FileIPAuth` modules extended to support prefixes.
- Firebird/Interbase driver for SQL modules.
- NAT support for unregistred callers.
- NumberAnalysis routing policy extended to support Setup messages.
- New `CapacityControl` module for controlling inbound traffic.
- Default `SignalTimeout` value increased from 15 to 30 seconds.
- New `StatusAcct` accounting module for the status port sponsored by Grupo Isec.

The version 2.2.3 contains the following new features and bugfixes:

- New `[RewriteCLI]` options to control CLIR features precisely.
- New CLIR/CLIP (Calling Line Identification Restriction/Presentation) features in the `[RewriteCLI]` module. Ability to hide CLI (enable CLIR/CLIP) per endpoint.
- New `[RoutedMode] SocketCleanupTimeout` config variable. More reliable socket management and better network error reporting.
- The `RewriteCLI` module rewrites outbound rules correctly for random numbers separated by commas.
- Broadcast request are handled correctly in the LARGE_FDSET mode.
- New ENUM routing policy from Simon Horne.
- NetMeeting compatibility problems fixed.
- A new `[RasSrv::RRQFeatures] IRQPollCount` config variable. Default number of "poll" IRQ messages changed from 2 to 1 to allow faster endpoint unregistration.
- Tunneled H.245 messages were not processed correctly (were ignored).
- 8.2 ([FileIPAuth]) module moved from the contrib section into the main branch.

The version 2.2.2 contains the following new features and bugfixes:

- New CLI rewrite types - prefix replacement (*=) and identity match (=).
- A new `TranslateFacility` config variable to enable Facility message conversion between H.323v4 and previous versions of the protocol.
- `SignalTimeout`, `AlertingTimeout` and %t, %p, %{ring-time}, %{alerting-time} accounting variables ported from 2.0 branch. `ConnectTimeout` config variable replaced with `SignalTimeout` and `AlertingTimeout`.

- A new `%r` accounting variable to provide information about who was the one that disconnected a call.
- A new, generic 8.7 (SQLAuth) module for RRQ, ARQ, LRQ and Setup authentication and authorization.
- New –core command line argument to enable core dump generation for Unix.
- A new `Vendor` config variable for 12.2 ([Endpoint]) section to provide vendor specific extensions when registering with a parent gatekeeper.
- LRQ nonStandardData field was not included for CiscoGK neighbors.
- New 6.6 ([RewriteCLI]) config section that allows arbitrary rewriting of ANI/CLI numbers.
- New 6.5 (numberanalysis) routing policy.
- New FileIPAuth module in the contrib/ipauth directory.
- Call accounting updates/call disconnect handling is now more robust and does not lock the whole call table and (effectively) the gatekeeper for long time periods.
- Do not support multiple rewrite targets, as this feature does not work well if rewrite is performed more than once.
- The gatekeeper could crash if the connection was closed before the welcome message has been sent to the client.
- Different Username was reported during Setup auth and acct step, if no sourceAddress has been present for an unregistered call.
- More missing config reload locks added to allow seamless config reload.
- A default value for the config variable `ForwardOnFacility` changed to `0`.
- Ability to encrypt all passwords in the config. A new `EncryptAllPasswords` config variable, `KeyFilled` config variable usage extended.
- Ability to read config settings from an SQL database ported from 2.0 branch. Read 12.4 ([SQLConfig]) for more details.
- Framed-IP-Address could not be determined for unregistered calls with no Setup-UUIE.sourceCallSignalAddress field, causing authentication to fail.
- Provide proper handling of aliases of type partyNumber (e164Number or privateNumber).
- A fix for RTP/Q931/H245/T120PortRange to correct a bug with port range wraparound if the last port is 65535. This caused a next port to be set to 0 and any subsequent port allocation to fail.
- Dynamic allocation of RTP ports did not work, use a fixed port range 1024-65535 as a default for the `RTPPortRange` config variable.
- Obsolete auth modules MySQLAliasAuth and MySQLPasswordAuth are now removed.
- SQL modules accept only one database host in the `Host` parameter.

The version 2.2.1 contains the following new features and bugfixes:

- Enhanced prefix matching for routing policies. A dot (`.`) matches any digit.

- Enhanced prefix matching for neighbors. A dot (.) matches any digit, ! at the beginning disables the prefix.
- A missing lock during config reload caused the gatekeeper to crash.
- More reliable port number selection for Q.931, H.245, T.120 and RTP port ranges (earlier, a config reload could cause many calls to fail because of inability to allocate a new socket).
- Default value for RTPPortRange is now to let the OS select a port number.
- More flexible rewrite rules (both global and per-gw) with new '.' and '%' wildcard characters.
- Enhanced prefix matching for gateways. A dot (.) matches any digit, ! at the beginning disables the prefix.
- Insert missing Calling-Party-Number-IE/Display-IE if corresponding `Screen...` options are enabled.
- Shutdown the gatekeeper if there are errors in SQL auth/acct modules configuration.
- Called-Station-Id number type can be selected between the original one (dialed number) and the rewritten one. New `UseDialedNumber` config option for 8.9 (RadAuth)/8.10 (RadAliasAuth) /9.3 (RadAcct) modules, new %{Dialed-Number} variable for 9.4 (SQLAcct) and 9.2 (FileAcct) modules.
- Ability to customize timestamp formats. New `TimestampFormat` config variables for main, 9.4 ([SqlAcct]), 9.3 ([RadAcct]), 9.2 ([FileAcct]) and 12.1 ([CallTable]) sections.
- RadAuth/RadAliasAuth modules can now add/remove endpoint aliases during endpoint registration (using h323-ivr-in=terminal-alias: Cisco AV-Pair).
- New `TcpKeepAlive` option to solve the problem with network errors and calls hanging in the call table. See docs/keepalive.txt for more details.
- New status port `RouteToGateway` command.

The version 2.2.0 contains the following new features and bugfixes:

- New `RoundRobinGateways` config option.
- Call capacity limits and priority routing for gateways. New `EP::` config sections for per-endpoint configuration settings (see 11 (Per-Endpoint Configuration Settings)).
- RTP proxy handling moved to a separate RTP proxy threads, so processing of signaling messages does not block RTP packets. New `RtpHandlerNumber` config option.
- REUSE_ADDRESS option enabled on listening sockets in non-LARGE_FDSET mode to fix a bug with the gatekeeper being unable to open listening ports after restart.
- Ability to set call destination in auth modules. RADIUS based call routing.
- Support for SqlBill tariff table import from an OpenOffice.org Calc spreadsheet.
- Fixed sourceInfo LRQ field handling - now it contains an H.323 identifier of the gatekeeper. Nonstandard data and gatekeeperIdentifier fields are set only when the neighbor is defined as GnuGk.
- Ability to set shared secrets for each radius server separately.
- New, much faster, Radius client implementation.
- Called-Party-Number-IE rewrite occurred too late, causing auth/acct modules to receive the original number instead of the rewritten one.

- Fixed proxying of RTP packets, so RTP sockets are not closed on temporary errors (like remote socket not yet ready). This bug affected especially NAT traversal and situation, when audio was sent very early, when reverse proxy path has not been yet established.
- Fixed handling of RRJ from an alternate GnuGk.
- New direct SQL accounting module (9.4 ([SQLAcct])).
- Handling multiple reply messages (RIP/LCF/LRJ) from neighbors fixed.
- Support for CallCreditServiceControl in RCF and ACF messages, which allows reporting call duration limit and user's account balance to endpoints. Currently RadAuth and RadAliasAuth modules support this feature.
- Log file rotation, new `LogFile` config section, new `setlog` and `rotatelog` status interface commands.
- Do not include an invalid access token (with null object identifier) in LCF to prevent interoperability problems.
- Better handling of multiple calls over a single signaling channel by setting multipleCalls and maintainConnection H.225.0 fields to FALSE in all messages passing through the gatekeeper.
- Better User-Name, Calling-Station-Id and Called-Station-Id handling.
- IncludeEndpointIP flag for RadAuth, RadAliasAuth and RadAcct is obsolete, these modules will always send Framed-IP-Address.
- New Gatekeeper::Auth flag SetupUnreg to toggle Q.931 Setup authentication for unregistered endpoints only.
- New RADIUS h323-ivr-out=h323-call-id parameter that contains an H.323 Call Identifier.
- The SQL billing from the contrib section can now authenticate users only by their IP (ignoring User-Name) and has a new, more flexible tariff/rating engine.
- RadAliasAuth can authenticate now Setup messages without sourceAddress field present (it will use Calling-Party-Number instead).
- Better signal handling to prevent accidental gatekeeper crashes (due to SIGPIPE, for example).
- CDR rotation per number of lines works correctly.

Of course, the major functions in version 2.0 are also included:

- The registration table and call record table are redesigned, thread-safe, and very efficient. Support ten thousands of registrations and thousands of concurrent calls.
- A new routed mode architecture that support H.225.0/Q.931 routed and H.245 routed without forking additional threads. Thus the thread number limit will not restrict the number of concurrent calls.
- Support H.323 proxy by routing all logical channels, including RTP/RTCP media channels and T.120 data channels. Logical channels opened by H.245 tunneling and fast-connect procedure are also supported. In proxy mode, there is no traffic between the calling and called parties 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.
- Support gatekeepers cluster by exchanging LRQ/LCF/LRJ (neighboring function). If the destination of a received LRQ is unknown, the GnuGk can forward it to next hop. Therefore the GnuGk can work as a directory gatekeeper.

- Support various authentication methods for selectable RAS requests, including H.235 password (MD5, SHA-1 and CAT), IP pattern and prefixes matching.
- Support alternate gatekeepers for redundancy and load balancing. If the GnuGk is overloaded, the endpoints can be redirected to other gatekeepers.
- Can work as an endpoint (gateway or terminal) by registering with a parent gatekeeper. With this feature, building gatekeeper hierarchies is easily.
- Monitor and control the GnuGk via TCP status port, including registration and call statistics.
- Output CDR(call detail record) to status port for backend billing system. The CDR contains call identifier, calling and called IP, start and end time and call duration.
- Most configurations are changeable at runtime. The GnuGk rereads the configurations on receiving `reload` command via status port, or on receiving `HUP` signal (Unix platform).

1.5 Download

The newest stable and a development version are available at *the download page* <http://www.gnugk.org/h323download.html>.

The very latest source code is in the CVS at *Sourceforge* <http://sourceforge.net/cvs/?group_id=4797> (*Web-GUI* <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/openh323gk/>). Beware - that's the bleeding edge.

You can also download some executables from *the download page* <http://www.gnugk.org/h323download.html>.

1.6 Mailing Lists

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

General user questions should be send to the *users mailing list* <mailto:Openh323gk-users@sourceforge.net>. You can find the list archive *here* <http://sourceforge.net/mailarchive/forum.php?forum_id=8549>. To join this mailing list, click *here* <http://lists.sourceforge.net/lists/listinfo/openh323gk-users>.

To report problems or submit bugs/patches, send mails to the *developers mailing list* <mailto:Openh323gk-developer@sourceforge.net>. The list archive is *here* <http://sourceforge.net/mailarchive/forum.php?forum_id=3079>. Please send user questions to the users mailing list and keep this list to development! If you want to contribute to the project, please *join the mailing list* <http://lists.sourceforge.net/lists/listinfo/openh323gk-developer>.

Note: Please don't send your questions as private emails to individual developer. We are usually busy. We would **not** like to be your private consultant, unless you'd like to pay us. Send your problems to the appropriate public mailing list so everybody can help you.

Also please don't send the GnuGk specific problems to the OpenH323 mailing list, or vice verse. They are different projects, though closely related.

Before you sending an email, make sure you have read the related documents carefully. Describe your problems clearly and precisely. Show us the error messages or logs if there is any.

1.7 Contributors

The current project coordinator is *Jan Willamowius* <http://www.willamowius.de/> <*jan@willamowius.de*>

The main features and functions of version 2.0 are contributed by Chih-Wei Huang <*cwhuang@linux.org.tw*> and *Citron Network Inc.* <http://www.citron.com.tw/>, including thread-safe registration and call tables, new routed mode architecture, H.323 proxy, H.235 authentication and MySQL backend.

Michal Zygmuntowicz <*m.zygmuntowicz@onet.pl*> has done some great work on Radius support and other improvements.

The initial version of the gatekeeper has been developed by Xiang Ping Chen, Joe Metzger and Rajat Todi.

2 Compiling and Installing

2.1 Compiling the Gatekeeper

To build the gatekeeper you need at least PWLib 1.5.0 and OpenH323 1.12.0 or later. The development version of the gatekeeper usually needs the most recent OpenH323 version available. These libraries are available at *OpenH323 Download Page* <http://sourceforge.net/projects/openh323>. See the instructions on *how to compile the OpenH323 code* <http://www.voxgratia.org/docs/faq.html>.

Order of compiling:

1. PWLib (release + debug version)
2. OpenH323
3. OpenH323 test application (not needed, just to make sure everything works so far)
4. The Gatekeeper

On Unix do a `configure` and `make debug` or `make opt` in the gatekeeper directory to build debug or release version, respectively. Use `make both` to build both versions. Note you have to use GCC 3.3.x or later. The older version may not work. Good practice is to do a `make debugdepend` or `make optdepend` in the gatekeeper directory before starting actual compilation (make debug or make opt) - these commands build appropriate dependency lists, so after you will update your sources from CVS, all affected files will get recompiled. Otherwise you can finish with the Gatekeeper partially compiled with the older headers and partially with the updated headers - a very bad thing.

On Windows just open and compile the provided solution (`gk.sln`) for Microsoft Visual Studio .NET 2003 or the workspace (`gk.dsw`) for Microsoft Visual Studio 6.0 SP6. Of course, you need to have PWLib and OpenH323 already compiled and appropriate include/library paths set up. If you want to get MySQL or PostgreSQL support, install/compile appropriate client libraries and add HAS_MYSQL=1 and/or HAS_PGSQL=1 to preprocessor macros of the gatekeeper project. You also need to tell the compiler where to find include files/libraries and instruct the linker to link with these client libraries.

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

The recommended PWLib/OpenH323 versions are these from the Pandora release (1.7.5.2/1.14.4.2) or newer. Older versions are not supported anymore and are not guaranteed to work with the gatekeeper.

To build the gatekeeper that is statically linked with system and OpenH323 libraries, `make optnoshared` or `make debugnoshared` has to be used.

In order to use the gatekeeper under heavy load, enabling LARGE_FDSET feature (ONLY FOR UNIX VERSION) 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. Note that the PWLib library starting from version 1.5.3 supports a similar feature too, so you can choose between LARGE_FDSET GnuGk and PWLib implementation. GnuGk native implementation gives better performance results. Maximum LARGE_FDSET value should be calculated based upon predicted maximum sockets usage. A rule of thumb may be:

```
MAX_NUMBER_OF_CONCURRENT_CALLS * 10 * 120%

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

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

2.2 The addpasswd Utility

Status line access authentication and SimplePasswordAuth module require encrypted passwords to be stored in the gatekeeper configuration file. Also since 2.2.2 version, the gatekeeper supports encryption of all passwords in the config. `addpasswd` utility is required to generate and store these encrypted passwords. This utility is included with the gatekeeper and can be compiled using:

```
$ make addpasswd
```

The usage is as follows:

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

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

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

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

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

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

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

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

2.3 Installing the Gatekeeper

There is no special installation procedure needed. Just copy the executable to the directory you like and create a config file for it. There are several config examples in the `etc/` subdirectory of source tree. See section 4.2 (Configuration File) for detailed explanations.

For example, in Linux x86 platform, the optimized executable `gnugk` is produced in `obj_linux_x86_r/` subdirectory. You may copy it to `/usr/sbin/`, create a config in `/etc/gnugk.ini` and start it by

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

See section 4.1 (Command Line Options) for details.

2.4 Pre-Built Binaries

If you do not wish to compile the gatekeeper from source, there are several pre-built 'packages' available from *SourceForge* <http://sourceforge.net/project/showfiles.php?group_id=4797>. Not all versions will be made available as binaries - check what is available.

Tar packages (.tgz or .tar.gz)

Download the tar file and enter the following command as `root`, substitute in the name of the file you downloaded.

```
$ tar xvzf gnugk-x.x.x.tar
```

Debian packages (.deb)

Debian includes the gatekeeper as openh323gk package. You can install it by using the following command as `root`:

```
$ apt-get install openh323gk
```

3 Getting Started (Tutorial)

3.1 A first simple experiment

To see that all components are up and running, get 2 Linux workstations, both connected to the LAN. Make sure you have at least version 1.1 of OpenH323 and OhPhone installed. On the first machine run the gatekeeper and ohphone (on different consoles):

```
jan@machine1 > gnugk -ttt
```

Now the gatekeeper is running in direct mode. The "`-ttt`" option tells the gatekeeper to do a lot of debug output on the console (you can direct that output to a file with "`-o logfile`").

```
jan@machine1 > ohphone -l -a -u jan
```

Now this OhPhone is listening (`-l`) for calls and will automatically accept them (`-a`). It has registered as user jan with the gatekeeper that it will automatically detect. (If the auto detect fails for some reason use "`-g 1.2.3.4`" to specify the IP number the gatekeeper is running on.)

On the second machine run ohphone only:

```
peter@machine2 > ohphone -u peter jan
```

The second instance of OhPhone registers with the auto detected gatekeeper as user peter and tries to call user jan. The gatekeeper will resolve the username to the IP number from where user jan has registered (machine1 in this case) and OhPhone will call the other instance of OhPhone on machine one.

The first instance of OhPhone will accept that call and Peter and Jan can chat.

3.2 Using the Status interface to monitor the gatekeeper

Now we try to see which messages are handled by the gatekeeper. On a new console on machine1 we use telnet to connect to the gatekeeper:

```
jan@machine1 > telnet machine1 7000
```

Most probably we'll get an "Access forbidden!" message, because not everybody is allowed to spy.

Now we create a file called `gatekeeper.ini` and put it in the directory where we start the gatekeeper. `gatekeeper.ini` only contains 4 lines:

```
[Gatekeeper::Main]
Fortytwo=42
[GkStatus::Auth]
rule=allow
```

Stop the gatekeeper with Ctrl-C and restart it. When we do the telnet again, we stay connected with the gatekeeper. Now repeat the first experiment where Peter calls Jan and see which messages are handled by the gatekeeper in non-routed mode. There is a number of commands that can be issued in this telnet session: Type "help" to see them. To end the telnet session with the gatekeeper type "quit" and hit Enter.

But this is very insecure - everybody could connect to the status interface and see this. Lets change the configuration file to

```
[Gatekeeper::Main]
Fortytwo=42
[GkStatus::Auth]
rule=password
gkadmin=QC7VyAo5jEw=
```

The 5th line is added by addpasswd utility, it creates a user "gkadmin" with password "secret" which will limit access to the status port. Restart the gatekeeper with this new configuration and do the telnet again. Now you will be asked for a username and password before you can login.

Take a look at the 4.2.2 (GkStatus::Auth) section for more details on securing the status ports.

3.3 Starting the gatekeeper in routed mode

Starting the gatekeeper in routed mode means that the gatekeeper uses "gatekeeper routed signaling" for all calls. In this mode the gatekeeper all signaling messages go through the gatekeeper and it has much greater control over the calls.

```
jan@machine1 > gnugk -r
```

Now the gatekeeper is running 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 ohphone).

3.4 A virtual PBX: Disconnecting calls

Until now the gatekeeper has only acted as a mechanism to resolve symbolic names to IP addresses. That's an important function but hardly exciting.

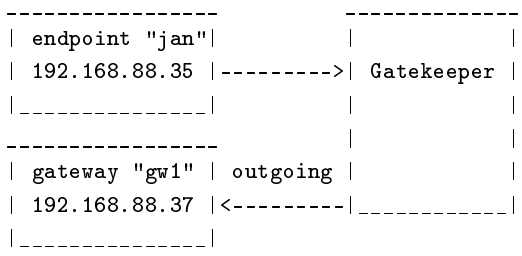
Since the gatekeeper has a lot of control over the calls, it can terminate them for example. When we are connected to the status port, we can list all active calls with "`PrintCurrentCalls`". To terminate a call, we can say "`Disconnectip 1.2.3.4`" for one of its endpoints.

One could for example write a simple script that connects to the status port and listens for all ongoing calls and terminates them after 5 minutes, so no user can over use system resources.

Take a look at the other telephony functions like TransferCall to see what else is available.

3.5 Routing calls over 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.



The gatekeeper has to know which calls are supposed to be routed over the gateway and what numbers shall be called directly. Use the [RasSrv::GWPrefixes] section of the config file to tell the gatekeeper the prefix of numbers that shall be routed over 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 that you must use the gateway alias - you can't just tell the gatekeeper the IP number 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 accordingly to the longest matching prefix rule, with ! rules having higher priority if lengths are equal. Some examples:

```
[RasSrv::GWPrefixes]
; This entry will route numbers starting with 0048 (but not with 004850 and 004860)
; to the gw1
gw1=0048,!004850,!004860
```

```
; This entry will match only 001 with 10 digits following
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 6.2 (RasSrv::RewriteE164) section to configure that.

Example: You want to call number 12345 with you 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 6.3 (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 at both left and right hand sides of expressions, the percent sign can occur only at the left side only. 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
```

4 Basic Gatekeeper Configuration

The behavior of the gatekeeper is completely determined by the command line options and configuration file. Some command line options may override the setting of 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.

`-s -section section`

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

`-i -interface IP`

Specify the interface (IP number) that the gatekeeper listens to. You should leave out this option to let the gatekeeper automatically determine the IP it listens to, unless you want the gatekeeper only binds to a specified IP.

`-l -timetolive n`

Specify the time-to-live timer (in seconds) for endpoint registration. It overrides the setting `TimeToLive` in the configuration file. See 4.2.1 (there) for detailed explanations.

`-b -bandwidth n`

Specify the total bandwidth available for the gatekeeper. Without specifying this option, the bandwidth management is disable by default.

`-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 versions.

`-core n`

(Unix only) 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.

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. The more `-t` you add, the more verbose to output. For example, use `-ttttt` to set the trace level to 5.

4.2 Configuration File

The 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 the 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 runtime. Once you modify the configuration file, you may issue `reload` command via status port, or send a signal `HUP` to the gatekeeper process on Unix. For example,

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

4.2.1 Section [Gatekeeper::Main]

- `Fortytwo=42`
  Default: `N/A`
  This setting is used to test the presence of the config file. If it is not found, a warning is issued. Make sure it's in all your config files.

- `Name=OpenH323GK`
  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.

- `Home=192.168.1.1`
  Default: `0.0.0.0`
  The gatekeeper will listen for requests on this IP number. By default, the gatekeeper listens on all interfaces of your host. You should leave out this option, unless you want the gatekeeper only to bind to a specified IP. Multiple Home addresses can be used and have to 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 detect the interfaces of your host automatically. There are two situations that you may want to use this option. One is automatically detection failed, another is the gatekeeper is behind a NAT box and allow endpoints with public IPs to register with. In this case you should set the option just as the gatekeeper is running on the NAT box.

- `EndpointIDSuffix=_gk1`
  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 is only useful when using more than one gatekeeper.

- `TimeToLive=300`
  Default: `-1`
  An endpoint's registration with a gatekeeper may have a limited life span. The gatekeeper specifies the registration duration of an endpoint by including a **timeToLive** field in the RCF message. After the specified time, the registration has expired. The endpoint shall periodically send an 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. This is called a lightweight RRQ.

  This configuration setting specifies the time-to-live timer in seconds until the registration expires. Note the endpoint may request a shorter **timeToLive** in the RRQ message to the gatekeeper. To avoid an overload of RRQ messages, the gatekeeper automatically adjusts this timer to 60 seconds if you give a lesser value!

  After the expiration time, the gatekeeper will subsequently send two IRQ messages to query if the endpoint is still alive. If the endpoint responds with an IRR, the registration will be extended. Otherwise 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`.

- `TotalBandwidth=100000`
  Default: `-1`
  Total bandwidth available to be given to endpoints. By default this feature is off. Be careful when using it, because many endpoints have buggy implementations.

- `RedirectGK=Endpoints > 100 || Calls > 50`
  Default: `N/A`
  This option allow you to redirect endpoints to alternate gatekeepers when the gatekeeper overloaded. For example, with the above setting the gatekeeper will reject an RRQ if registered endpoints 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 return an 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. Please also note the function only takes effect to H.323 version 4 compliant endpoints.

- `AlternateGKs=1.2.3.4:1719:false:120:OpenH323GK`
  Default: `N/A`
  We allow for existence of another gatekeeper to provide redundancy. This is implemented in a active-active manner. Actually, you might get into a (valid !) situation where some endpoints are registered with the first and some are registered with the second gatekeeper. You should even be able use the two gatekeepers in a round_robin fashion for load-sharing (that's untested, though :-)). If you read on, "primary GK" refers to the gatekeeper you're currently configuring and "alternate GK" means the other one. The primary GK includes a field in the RCF to tell endpoints which alternate IP and gatekeeper identifier to use. But the alternate GK needs to know about every registration with the primary GK or else it would reject calls. Therefore our gatekeeper can forward every RRQ to an alternate IP address.

  The AlternateGKs config option specifies the fields contained in the primary GK'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 GK before placing calls. They usually don't because we forward their RRQs, so they get registered with the alternate GK, too. The fourth field specified the priority for this GK. Lower is better, usually the primary GK is considered to have priority 1. The last field specifies the alternate gatekeeper's identifier.

- `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, so it's a separate config option (think of multihomed machines).

- `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 GK (this statement is true for both, primary and alternate GK). Two mechanisms are used to identify whether a request should be forwarded. The first one looks for a flag in RRQ. Since few endpoints implement this, we need a second, more reliable way. 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.

- `SignalCallId=1`
  Default: `0`
  Signal call IDs in ACF/ARJ/DCF/DRJ/RouteRequest messages on the status port. See 13 (this section) for details.

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

- `TimestampFormat=ISO8601`
  Default: `Cisco`
  Control default format of timestamp strings generated by the gatekeeper. This option affects 9.4 ([SqlAcct]), 9.3 ([RadAcct]), 9.2 ([FileAcct]) and other modules, except 12.1 ([CallTable]). You can further customize timestamp formatting per-module by configuring per-module `TimestampFormat` setting.

  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 you need another format, you can build your own format string, using rules known from `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 have to be encrypted using `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` inside 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 instances of gatekeepers on the same machine you should turn this off.

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

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

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

- `EndpointSignalPort=1720`
  Default: `1720`
  Default port for call signaling channel of endpoints.

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

- `SignalReadTimeout=1000`
  Default: `1000`
  Time in milliseconds for read timeout on call signaling channels (Q931).

- `StatusReadTimeout=3000`
  Default: `3000`
  Time in milliseconds for read timeout on status channel.

- `StatusWriteTimeout=5000`
  Default: `5000`
  Time in milliseconds for write timeout on status channel.

4.2.2 Section [GkStatus::Auth]

Define a number of rules who is allowed to connect to the status port. Whoever has access to the status port has full control over your gatekeeper. Make sure this is set correctly.

- `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 peering 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 has to input appropriate username and password to login. The format of username/password is the same as 8.3 ([SimplePasswordAuth]) section.

  Moreover, these rules can be combined by "|" or "&". For example,

  - `rule=explicit | regex`
    The IP of client must match `explicit` **or** `regex` rule.
  - `rule=regex & password`
    The IP of client must match `regex` rule, **and** the user has to login by username and password.

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

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

- `DelayReject=5`
  Default: `0`
  How long (in seconds) to wait before rejecting invalid username/password for the status line access.

4.2.3 Section [LogFile]

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

- `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 the rotation, do not set `Rotate` parameter or set it to 0.

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

```
[LogFile]
Rotate=Hourly
RotateTime=45
```

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

```
[LogFile]
Rotate=Daily
RotateTime=23:00
```

  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 passwd in two ways. The first method is Direct Endpoint Call Signaling, in which case the call signaling messages are passed directly between the endpoints. The second method is Gatekeeper Routed Call Signaling. In this method, the call signaling messages are routed through the gatekeeper between the endpoints. The choice of which methods is used is made by the gatekeeper.

When Gatekeeper Routed call signaling is used, the gatekeeper may choose whether to route the H.245 control channel and logical channels.

Case I.

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

Case II.

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

Case III.

The gatekeeper routes the H.245 control channel, as well as all logical 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 an H.323 Proxy, which can be regarded as an 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). All settings in this section are affected by reloading.

- `GKRouted=1`
  Default: `0`
  Whether to enable the gatekeeper routed signaling mode.

- `H245Routed=1`
  Default: `0`
  Whether to route the H.245 control channel through the gatekeeper too. Only takes effect if `GKRouted=1` and H.245 tunneling is disabled for a call. Even when this option is disabled, if Proxy or ProxyForNAT takes effect, an H.245 channel is always routed through the gatekeeper for calls being proxied.

- `CallSignalPort=0`
  Default: `1721`
  The port of call signaling for the gatekeeper. The default port is `1721`. We don't use the well-known port `1720` so you can run an H.323 endpoint in the same machine of the gatekeeper. You may set it to `0` to let the gatekeeper choose an arbitrary port.

- `CallSignalHandlerNumber=2`
  Default: `1`
  The number of threads dedicated to handle signaling/H.245 channels. 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. A special care has to be taken on Windows, at RTP handling threads are subject of 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 its neighbors or parent GK. The gatekeeper will also use it's own call signaling address in LCF in responding to an LRQ. That means, the call signaling will be routed to GK2 in GK-GK calls. As a result, the CDRs in GK2 can 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. However, it raises security risks. Be careful to use it.

- `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.

- `RemoveCallOnDRQ=0`
  Default: `1`
  With this option turning off, 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 callee pickups. Setting this parameter to 1 makes the gatekeeper always send Release Complete to both endpoints before closing the call when it receives DRQ from one of the parties.

- `SupportNATedEndpoints=1`
  Default: `0`
  Whether to allow an endpoint behind an 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.

  Since 2.0.2, the 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 the GnuGk and you can make call now.

- `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**, 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**, 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.

- `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 your range is 2000-2001 and you made two calls, next two 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=0`
  Default: `1`
  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.

- `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.

- `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. Currently only routes available via the 'internal' routing policy can be used as alternatives routes.

  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.

- `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).

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`
  Define the networks behind the proxy. Multiple internal networks are allow. The proxy route channels only of the communications between one endpoint in the internal network and one external. If you don't specify it, all calls will be proxied.

  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`

- `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. Note the range size may limit the number of concurrent calls.

- `ProxyForNAT=1`
  Default: `1`
  If yes, the gatekeeper will proxy for calls to which one of the endpoints participated is behind an NAT box. This ensure 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 have better to disable this feature and let the NAT box forward RTP/RTCP stream for you.

- `ProxyForSameNAT=0`
  Default: `0`
  Whether to proxy for calls between endpoints from the same NAT box. You do not need to enable this feature in general, since usually endpoints from the same NAT box can communicate with each other.

- `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.

6 Routing Configuration

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

6.1 Section [RoutingPolicy]

This section explains how the various possible routing policies within the gatekeeper work.

The incoming call requests can be routed using a number of routing providers:

- `explicit`
  The destination is explicitly specified in the routing request.

- `internal`
  The classical rule; search the destination in RegistrationTable

- `parent`
  Route the call using information sent by the parent GK in reply to an ARQ the gatekeeper will send.

- `neighbor`
  Route the call using neighbors by exchanging LRQ messages

- `dns`
  The destination is resolved from DNS, provided it is resolvable

- `vqueue`
  Use the virtual queue mechanism and generate a RouteRequest event to let an external application do the routing (can only be used OnARQ)

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

- `enum`
  ENUM (RFC3761) is a method to use DNS lookup to convert real IDD E164 numbers into H323 dialing information. The servers it looks up by default are `e164.voxgratia.net`, `e164.org` and `e164.arpa`. To specify your own server you have to specify an environmental variable PWLIB_ENUM_PATH with the address of your preferred enum servers separated by a semicolon (;). (PWLIB_ENUM_PATH is supported starting with PWLib 1.8.0; 1.7.5.2 (Pandora) doesn't support it.)

  The enum policy replaces the destination with the information returned by ENUM server, so you must have the appropriate routing policies to finally route the call after the enum policy. Usually you should

also have the dns policy after the enum policy, since the new location is often returned in the form of 'number@gatekeeper' and the dns policy is 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.

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 kind of a default for the other types.

Example:

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

When one of the messages is received which calls for 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, first the neighbors are checked and 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 that fails the parent.

For the ARQ, LRQ, Setup and Facility messages one would use the [RoutingPolicy::OnARQ], [RoutingPolicy::OnLRQ], [RoutingPolicy::OnSetup] and [RoutingPolicy::OnFacility] sections using the 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,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 is beginning 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::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. In combination with the 6.2 (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:

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

If the call matches the gateway, 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. Multiple rules for the same gateway should be separated by ';'.

Example:

`gw1=in=123=321`

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

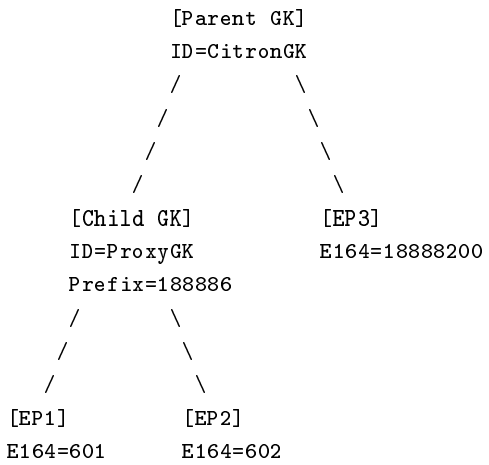
6.4 Section [Endpoint::RewriteE164]

Once you specify prefix(es) for your gatekeeper endpoint, the parent gatekeeper will route calls with **dialedDigits** 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,



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 GK 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 6.2 (RasSrv::RewriteE164), though the later will take effect first.

6.5 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 - 10 digits and to all other at least 6 and at most 20 digits.

6.6 Section [RewriteCLI]

This section contains a set of rewrite rules for ANI/CLI numbers (caller id). The rewrite process is done at two stages - inbound rewrite and outbound rewrite. The inbound rewrite is done before any other Q.931 Setup message processing (like inbound GWRewrite, authentication, accounting, ...) and it will have visible effect inside auth/acct modules, as it affects Calling-Station-Id. 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).

- `ProcessSourceInfo=1`
  Default: `1`
  In addition to rewriting a Calling-Party-Number IE also a sourceInfo 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`
  Here a global presentation indicator 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 send 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 tells that this is an inbound rule and the `CALLER_IP` will be used to match the rule (it can be a single IP or a whole subnet).

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 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 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 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, there follows a list of new CLI values to be used. If more than one value is specified, a 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", that can be used in place of the `CALLER_IP` or the `number_prefix`. To enable `CLIR` for this rule, use a 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 tell 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:any=any=0
```

These rules tell 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 other calls, set CLI to 0.

Example 3:

```
[RewriteCLI]
%r1% in:192.168.1.0/24=0048*=48
%r2% in:192.168.1.0/24=0*=48
in:any=100.~=48900900900
```

These rules tell 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 whole network address.

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 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 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 follows. If more than one value is specified, a one will be chosen on a random basis. It's possible to specify whole number ranges, like 49173600000-49173699999. There is a special string constant "any", that 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
```

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.

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 only a 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 actually, 2) force CLI presentation for other endpoints.

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 Windows platform, there is a problem with duplicated config keys, so there is 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 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
```

7 RAS Configuration

7.1 Section [RasSrv::GWPrefixes]

This section lists what E.164 numbers are routed to a specific gateway.

Format:

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

Note you have to specify the alias of the gateway. If a gateway registered with the alias, all numbers beginning with the prefixes are routed to this gateway. Special characters `.` and `!` can be used here to match any digit and disable the prefix.

Example:

`test-gw=02,03`

7.2 Section [RasSrv::PermanentEndpoints]

In this section you can put endpoints that don't have RAS support or that you don't want to be expired. The records will always keep in registration table of the gatekeeper. However, You can still unregister it via status port. Special characters `.` and `!` can be used with prefixes here to match any digit and disable the prefix.

Format:

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

Example:

For gateway,

```
10.0.1.5=Citron;009,008
```

For terminal,

```
10.0.1.10:1720=700
```

7.3 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 containing **supportedPrefixes** in the **terminalType** field of RRQ. This option defines whether to accept the specified prefixes of a gateway.

- `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 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.

7.4 Section [RasSrv::ARQFeatures]

- `ArjReasonRouteCallToSCN=0`
  Default: `1`
  If yes, the gatekeeper rejects a call from a gateway to itself by reason **routeCallToSCN**.

- `ArjReasonRouteCallToGatekeeper=1`
  Default: `1`
  If yes, 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.

- `CallUnregisteredEndpoints=0`
  Default: `1`
  With this option set on, the gatekeeper will accept an ARQ from a registered endpoint with **destCallSignalAddress**, no matter the address is belongs to a registered endpoint or not. That means you can explicitly specify the IP of endpoint (registered or not) you want to call.

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

7.5 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 an endpoint with alias 601 is behind NAT.

```
601=true
```

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 for the gatekeeper.

Syntax:

```
authrule=actions

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

A rule may results in one of the three codes: ok, fail, pass.

- `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, it is passed to next rule.
- `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.

Currently supported modules:

- `SimplePasswordAuth/SQLPasswordAuth` These modules check the **tokens** or **cryptoTokens** fields of 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 (libssl must be installed!) are supported now. For **tokens** tokens hashed by CAT (Cisco Access Token) and a clear text username/password are supported now. The ID and password are read from 8.3 ([SimplePasswordAuth]) section, an SQL database for `SimplePasswordAuth` and `SQLPasswordAuth` modules. `MySQLPasswordAuth` module is supported for backward compatibility.
- `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 8.5 ([RasSrv::RRQAuth]) section. For `SQLAliasAuth`, the pattern is retrieved from an SQL database, defined in 8.6 ([SQLAliasAuth]) section.
- `FileIPAuth` This module provides a simple way to restrict access to the gatekeeper based on caller's IP/network.
- `PrefixAuth` The IP or aliases of a request with a given prefix must match a specified pattern. See section 8.8 ([PrefixAuth]) for details. Currently the module can only authorize AdmissionRequest (ARQ) and LocationRequest (LRQ).
- `RadAuth` Provides authentication based on H.235 username/password security scheme. Authenticates RRQ, ARQ and Q.931 Setup through remote RADIUS servers. It passes to RADIUS servers usernames and passwords extracted from CAT (Cisco Access Tokens) **tokens** carried inside RRQ, ARQ or Setup packets. Therefore if your endpoints do not support CATs or you do not need authentication scheme based on individually assigned usernames/password - this module will not work for you (but you may check `RadAliasAuth` module). See section 8.9 ([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 8.10 ([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, like 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 8.7 ([SQLAuth]) for more details.
- `CapacityControl` A flexible module to control inbound call volume with ability to configure various conditions. IMPORTANT: It has to be used in conjunction with `CapacityControl` accounting module. See section 8.11 ([CapacityControl]) for more details.

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

Example 1:

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

The example below authenticates all calls, checking signaling Setup message details, using 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=optional;RRQ,ARQ
RadAliasAuth=required;RRQ,ARQ,SetupUnreg
default=allow
```

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`, `LRQ`, `Setup` and `SetupUnreg`. Format of a single entry is:

`IP=[allow | reject][;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 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.

Example #1:

```
[Gatekeeper::Auth]
FileIPAuth=required;RRQ,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
any=reject
```

Example #2:

```
[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
```

8.3 Section [SimplePasswordAuth]

The 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
```

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` and all its descendants will cache an authenticated password. This field define the cache timeout value in second. `0` means never cache the password, while a negative value means the cache never expires.

8.4 Section [SQLPasswordAuth]

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

- `Driver=MySQL | PostgreSQL | Firebird`
  Default: `N/A`
  SQL database driver to use. Currently, `MySQL`, `PostgreSQL` and `Firebird` drivers are implemented.
- `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`.
- `Database=billing`
  Default: `billing`
  The database name to connect to.
- `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 is in an encrypted form and should be created using the `addpasswd` utility.
- `CacheTimeout=120`
  Default: `0`

This field defines how long (alias;password) pairs retrieved from the database will be cached in the local memory. 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).

- `MinPoolSize=5`
  Default: `1`
  Define the number of active SQL connections. This allows better performance under heavy load, because more than 1 concurrent query can be executed at the same time. `MinPoolSize=1` setting simulates old behavior, when access to the SQL database is serialized (one query at time).

- `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 [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.

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 using (commonly used) decimal notation: "`byteA.byteB.byteC.byteD:port`". Example:

```
sigip:192.168.242.165:1720
```

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

8.6 Section [SQLAliasAuth]

Authenticate endpoints using rules stored in the SQL database (the rules conform to the format defined in the 8.5 ([RasSrv::RRQAuth]) section). This section defines SQL driver to use, SQL database connection parameters and the query to use to retrieve the patterns.

- `Driver=MySQL | PostgreSQL | Firebird`
  Default: `N/A`
  SQL database driver to use. Currently, `MySQL`, `PostgreSQL` and `Firebird` drivers are implemented.

- `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`.

- `Database=billing`
  Default: `billing`
  The database name to connect to.

- `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 is in encrypted form and should be created using the `addpasswd` utility.

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

- `MinPoolSize=5`
  Default: `1`
  Define the number of active SQL connections. This allows better performance under heave load, because more than 1 concurrent query can be executed at the same time. `MinPoolSize=1` setting simulates old behavior, when access to the SQL database is serialized (one query at time).

- `Query=SELECT ...`
  Default: `N/A`
  Defines 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'
```

8.7 Section [SQLAuth]

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

- `Driver=MySQL | PostgreSQL | Firebird`
  Default: `N/A`
  SQL database driver to use. Currently, `MySQL`, `PostgreSQL` and `Firebird` drivers are implemented.
- `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`.
- `Database=billing`
  Default: `billing`
  The database name to connect to.
- `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 is in encrypted form and should be created using the `addpasswd` utility.
- `MinPoolSize=5`
  Default: `1`
  Define the number of active SQL connections. This allows better performance under heave load, because more than 1 concurrent query can be executed at the same time. `MinPoolSize=1` setting simulates old behavior, when access to the SQL database is serialized (one query at time).
- `RegQuery=SELECT ...`
  Default: `N/A`
  Define an 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 an 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

  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 authentication result and 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 authenticated successfully, 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, aliase
```

- `NbQuery=SELECT ...`
  Default: `N/A`
  Define an 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 authentication result and 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 an 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 - an 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)
- %{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 authentication result and 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
   (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)
   (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

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
```

8.8 Section [PrefixAuth]

The section defines the authentication rule for `PrefixAuth` module. Currently, only ARQs and LRQs can be authorized by this module.

First, a 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 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 next authentication module according to the module requirement.

Format:

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

Syntax:

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

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

Where <iprule> can be specified in decimal dot notation or CIDR notation, <aliasrule> is expressed in regular expression. If the '!' flag precedes the rule, the sense is inverted.

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.*
ALL=allow ipv4:ALL
```

In this configuration, all endpoints except from network 10.0.0.0/27 are allow 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. All other situations are allowed.

8.9 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.

  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
  ```

- LocalInterface=IP_OR_FQDN
  Default: N/A
  Particular local network interface that RADIUS client 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 GK load time 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: `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 Section [RadAliasAuth]

This section defines configuration settings that enable RADIUS authentication based on endpoint aliases and/or IP addresses present in RRQ RAS requests, ARQ RAS request 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, 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`
  Particular local network interface that RADIUS client 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 GK load time 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;"
```

- `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 Section [CapacityControl]

This section contains a set of rules for controlling inbound call volume depending on various conditions. In order 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. `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.

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:

```
[RewriteCLI]
%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.

9 Accounting Configuration

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

9.1 Section [Gatekeeper::Acct]

The section defines a list of modules that will be performing accounting. The accounting is for logging gatekeeper on/off events and call start/stop/update events. Each accounting module logs received events to a module specific storage. Such storage can be a plain text file or a 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 | CapacityControl | ...
<actions> := <control>[;<event>,<event>,...]
<control> := optional | required | sufficient | alternative
<event>   := start | stop | connect | update | 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),
- `connect` - a call has been connected (only available in routed mode),
- `update` - a call is active and the periodic update is performed to reflect the new call duration. Frequency of such updates is determined by **AcctUpdateInterval** variable from 12.1 ([CallTable]) section,
- `stop` - a call has been disconnected (removed from the GK call table),
- `on` - the gatekeeper has been started,
- `off` - the gatekeeper has been shut down.

An event logging by a module may results in one of the 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. **control** flag for each module, along with result codes, define what is the final status of the event processing by the whole 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 immediatelly. 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 CDR text file logger. It outputs status line like CDR lines to a specified text file. This module supports only **stop** accounting event. Configuration settings are read from 9.2 ([FileAcct]) section.
- `RadAcct` This module performs RADIUS accounting. It supports all event types (start, stop, update, on, off). See the section 9.3 ([RadAcct]) for configuration details.
- `SQLAcct` This module performs direct SQL accounting. It supports (start, connect, stop, update) event types. See the section 9.4 ([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) event types. See the section 9.5 ([StatusAcct]) for configuration details.
- `CapacityControl` This module performs inbound call volume logging, required for `CapacityControl` authentication module to work correctly. See the section 8.11 ([CapacityControl]) for details.
- `default` This is a special pseudo module - it is used to set the final status if preceding modules have not determined it. The format is as follows:

  Syntax:

  ```
  default=<status>[;<event>,<event>,...]
  <status> := accept | fail
  <event>  := start | stop | update | on | off
  ```

The 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
```

The 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 returns **fail** return code, it is passed down to FileAcct module. FileAcct module does not support **start** events, so it returns **next** return code. If there were no the **default** rule, the final status would be failure, because no module has been able to log the event.

The 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 **fail** return code for **start** event, the code is passed down to FileAcct module. FileAcct module does not support **start** events, so it returns **next** return code. The **default** rule ensures, that the call is disconnected if call start event could not has been logged with RadAcct. But we 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 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 strings from the **CDRString** parametrized string.

- `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 how to build a custom CDRs. 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 a 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)
  - `%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)
  - %{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)
  - %{codec} - audio codec used during the call (only for H.245 routed/tunneled calls)

- `TimestampFormat=Cisco`
  Default: `N/A`
  Format of timestamp strings printed in CDR strings. If this setting is not specified, a global one from the main gatekeeper section is applied.

- `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). See the examples for more details.

  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
```

9.3 Section [RadAcct]

This accounting module sends accounting data to a RADIUS server. Module configuration is almost the same as for RADIUS authenticators (see 8.9 ([RadAuth]) and 8.10 ([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, port number from

`DefaultAcctPort` is be used. If no secret is set, the default shared secret from `SharedSecret` is used. Server names could be either IP addresses or DNS names.

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
```

- `LocalInterface=IP_OR_FQDN`
  Default: `N/A`
  Particular local network interface that RADIUS client should use in order to communicate with RADIUS servers.

- `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.

- `DefaultAcctPort=PORT_NO`
  Default: `1813`
  Default port number to be used for RADIUS accounting requests, if not overridden by `Servers` attribute.

- `SharedSecret=SECRET`
  Default: `N/A (empty string)`
  A 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.

- `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.

- `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, a 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 Section [SQLAcct]

This accounting module stores accounting information directly to an SQL database. Many configuration settings are common with other SQL modules.

- `Driver=MySQL | PostgreSQL | Firebird`
  Default: `N/A`
  SQL database driver to use. Currently, `MySQL`, `PostgreSQL` and `Firebird` drivers are implemented.

- `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`.

- `Database=billing`
  Default: `billing`
  The database name to connect to.

- `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 is in an encrypted form and should be created using the `addpasswd` utility.

- `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. Parameter placeholders are denoted by % character and can be one letter (like %u) or whole strings (like %{src-info}). Specify %% to embed a percent character inside the query string (like %%). For `SQLAcct` the following parameters are defined:

- %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 (**hexadecimal** integer)
- %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) call (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)
- %{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} - 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)
- %{codec} - audio codec used during the call (only for H.245 routed/tunneled calls)

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 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 SQL query used to update call record in the database with the current call state. 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 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 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)
  ```

- `TimestampFormat=MySQL`
  Default: `N/A`
  Format of timestamp strings used in queries. If this setting is not specified, a 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 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.

- `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 9.4 ([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, a global one from the main gatekeeper section is used.

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. But here is an example what those tables might look like in MySQL and you can use these 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.

```
GRANT delete,insert,select,update ON GNUGK.* TO 'YourDesiredUsername'@'localhost' IDENTIFIED BY 'APassword
```

With this command you will permit accessing the data from only the local server. If you need to access these data from any other computer then you have to set the proper security options.

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
```

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 "*" prefix. More than one prefix can be specified. You can use special characters "." and "!" to do wildcard matching and disable a specific prefix.

Conversely, the gatekeeper will only reply to LRQs sent from neighbors defined in this section. If you specify an empty prefix, no LRQ will be sent to that neighbor, but the gatekeeper will accept LRQs from it. By the empty prefix it is meant a single semicolon appended to the neighbor entry. Example:

```
GK1=192.168.0.5;
```

If you skip the semicolon, LRQs will be always sent to this neighbor.

The `password` field is used to authenticate LRQs from that neighbor. See section 8.1 ([Gatekeeper::Auth]) for details.

If a call from a neighbor is accepted also depends on the AcceptNeighborsCalls switch in the 5.1 ([RoutedMode]) section.

Neighbor handling has changed significantly from version 2.0 to version 2.2. Neighbors can be specified now the old or the new way.

Entry in the old format:

```
GKID=ip[:port;prefixes;password;dynamic]
```

Example:

```
GK1=192.168.0.5;*
GK2=10.0.1.1:1719;035,036;gk2
GK3=gk.citron.com.tw;;gk3;1
```

Entry in the new format:

```
GKID="GnuGK" | "CiscoGK" | "ClarentGK" | "GlonetGK"
```

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 new format specifies in the `[RasSrv::Neighbors]` section only the gatekeeper types. 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 responses from neighbors. If no response from all neighbors after timeout, the gatekeeper will reply an ARJ to the endpoint sending the ARQ.

- `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. This setting can be overridden with configuration of a particular neighbor.

- `AlwaysForwardLRQ=1`
  Default: `0`
  Force the gatekeeper to forward an LRQ even if there is no **hopCount** in the LRQ. To avoid LRQ loops, you should use this option very carefully. This option is used only for an old-style (2.0) neighbor configuration, the new one reads the settings from a neighbor-specific config section.

- `AcceptForwardedLRQ=1`
  Default: `1`
  Whether to accept an LRQ forwarded from neighbors. This setting can be overridden with configuration of a particular neighbor.

- `IncludeDestinationInfoInLCF=0`
  Default: `1`
  The gatekeeper replies LCFs containing **destinationInfo** and **destinationType** fields, the aliases and terminal type of the destination endpoint. The neighbor gatekeeper can then save the information to suppress later LRQs. However, some vendors' gatekeepers misuse the information, thus result in interoperability problems. Only turn off this option if you encounter problems upon communicating with a third-party gatekeeper.

- `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. 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.

10.2.1 Section [Neighbor::...]

Sections starting with `[Neighbor::` are for neighbor specific configuration.

- `GatekeeperIdentifier=GKID`
  Default: `N/A`
  Gatekeeper identifier for this neighbor. If this options is not specified, the identifier is taken from the second part of this `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 from incoming LRQs. `It is not yet implemented`.

- `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 LRQs to receive 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

- `AcceptPrefixes=*`
  Default: `*`
  A list of prefixes that the gatekeeper 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. This setting can be overridden with configuration of a particular neighbor.

11 Per-Endpoint Configuration

In addition to the standard configuration file options, per-endpoint configuration settings can be specified in the config file. The syntax is as follows:

11.1 Section [EP::...]

```
[EP::ALIAS]
Key Name=Value String
```

`ALIAS` is replaced with an actual alias for an endpoint the settings should apply to. 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 (that has available capacity).

- `GatewayPriority=1`
  Default: `1`
  Apply only to gateways. Allow priority based routing in case, when more than one gateway matches a dialed number. The smaller value the higher priority is assigned to a gateway. A call is routed to the first available gateway (that has available capacity) with the highest priority (the smallest `GatewayPriority` values).

- `GatewayPrefixes=0048,0049,0044`
  Default: `N/A`
  Additional prefixes for this gateway. Apply only to gateways. Special characters `.` and `!` can be used here to match any digit and disable the prefix.

- `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).

- `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).

Example:

```
[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. The IP and endpoint ID of the calling party is printed as empty. This is usually used for debug purpose.

- `GenerateUCCDR=0`
  Default: `0`
  Generate CDRs for calls that are unconnected. This is usually used for debug purpose. Note 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 calls 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 tells the gatekeeper to not send accounting updates at all. Please note that setting short periods may decrease GK performance.

- `TimestampFormat=Cisco`
  Default: `RFC822`
  Format of timestamp strings printed inside CDRs.

- `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 didn't 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.

12.2 Section [Endpoint]

The gatekeeper can work as an endpoint by registering with another gatekeeper. With this feature, you can easily build gatekeeper hierarchies. The section defines the endpoint features for the gatekeeper.

- `Gatekeeper=10.0.1.1`
  Default: `no`
  Define a parent gatekeeper for the endpoint(gatekeeper) to register with. Don't try to register with yourself, unless you want to be confusing. To disable this feature, set the field to be `no`.

- `Type=Gateway`
  Default: `Gateway`
  Define the terminal type for the endpoint. The valid values are `Gateway` or `Terminal`.

- `Vendor=Cisco | GnuGk | Generic`
  Default: `GnuGk`
  Choose parent gatekeeper type to enable vendor specific extensions.

- `H323ID=CitronProxy`
  Default: <Name>
  Specify the H.323 ID aliases for the endpoint. Multiple aliases can be separated by comma.

- `E164=18888600000,18888700000`
  Default: `N/A`
  Define the E.164 (dialedDigits) aliases for the endpoint. Multiple aliases can be separated by 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 both **cryptoTokens** and **tokens** fields, set the password to be empty. 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.

  Besides, the password is also 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 replied to the 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.

- `ARQTimeout=2`
  Default: `2`
  Define the timeout value in second for ARQs.

- `UnregisterOnReload=1`
  Default: `0`
  Defines whether the child gatekeeper unregisters and re-registers with it's parent when receiving a Reload command.

- `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 definite time only, it is good to set this to some value a bit shorter than NAT box mapping timeout. Refreshing is done by sending a special Q.931 IncomingCall-Proceeding message. If you NAT performs TCP port translation, you may need to set it to a values as short as 60 seconds.

- `Discovery=0`
  Default: `1`
  Decide whether to discover the parent gatekeeper by sending GRQ first.

- `UseAlternateGK=0`
  Default: `1`
  Enable alternate gatekeepers feature. If GRJ/GCF/RFC messages received from a parent gatekeeper contain a list of alternate gatekeepers, this information is stored and can be used to reregister with another gatekeeper in case of any failure. If you don't want to use this feature, set this variable to `0`.

- `GatekeeperIdentifier=ParentGK`
  Default: `Not set`
  Define it if you want to accept only such 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 set.

- `EndpointIdentifier=OpenH323GK`
  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.

12.3 Section [CTI::Agents]

This section allows the configuration of a so called virtual queues to allow inbound call distribution by an external application via the status port. A virtual queue has an H.323 alias that can be called like an endpoint.

Upon arrival of an ARQ on a virtual queue, the gatekeeper signals a RouteRequest on the status port and waits for an external application to respond with either a RouteReject (then the ARQ will be rejected) or with RouteToAlias/RouteToGateway which leads to the ARQ 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 an ARQ destination alias matches one these names, the virtual queue is activated,
- `prefix` - a list of prefixes is given. If an ARQ destination alias starts with one these prefixes, the virtual queue is activated,
- `regular expression` - a regular expression is given. If an ARQ destination alias matches the expression, the virtual queue is activated.

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]*$
  ```

- `RequestTimeout`
  Default: `10`
  Timeout in seconds for the external application to answer the RouteRequest. If no answer is received during this time an ARJ will be sent to the caller.

12.4 Section [SQLConfig]

Load gatekeeper settings from an 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.

- `Driver=MySQL | PostgreSQL | Firebird`
  Default: `N/A`
  SQL database driver to use. Currently, `MySQL`, `PostgreSQL` and `Firebird` drivers are implemented.

- `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`.

- `Database=billing`
  Default: `billing`
  The database name to connect to.

- `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 is in encrypted form and should be created using the `addpasswd` utility.

- `ConfigQuery=SELECT ...`
  Default: `N/A`
  Define an 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 into 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 an SQL query used to retrieve from the database rewrite rules for `[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'
```

- `NeighborsQuery=SELECT ...`
  Default: `N/A`
  Define an SQL query used to retrieve from the database neighbor entries for `[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'
  ```

- `PermanentEndpointsQuery=SELECT ...`
  Default: `N/A`
  Define an SQL query used to retrieve permanent endpoints from the database for `[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 four 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 (comma separated)

  Sample query strings:

  ```
  PermanentEndpointsQuery=SELECT peip, 1720, pealias, NULL FROM permanentep WHERE gk = '%1'
  ```

- `GWPrefixesQuery=SELECT ...`
  Default: `N/A`
  Define an SQL query used to retrieve gateway prefixes from the database for `[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'
```

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.

The messages sent by the gatekeeper to the status port are grouped into three **output trace levels**:

- 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), 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

  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 details 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 GUI for the Gatekeeper

There are several Graphical User Interface (GUI) frontends for the gatekeeper.

- Java GUIDeveloped 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 Java GUI* <http://www.gnugk.org/h323gui.html>

- GkGUIA new standalone Java program developed by *Citron Network Inc.* <http://www.citron.com.tw/> It requires Java 1.4. New features include:

  - Monitor multiple gatekeepers simultaneously.
  - Two view modes: Button List and Tree List.
  - Call Detail Record(CDR) and statistics.
  - GK Status Log.
  - Different colors for different endpoint types.
  - Modify gatekeeper configuration.
  - Forced unregister endpoints.
  - Save and print status log and CDR.

  The GkGUI is released under GNU General Public License, available at *GnuGk Development* <http://www.gnugk.org/h323develop.html#java>

13.2 Commands (Reference)

This section lists all commands that you can issue to the status port (manually or with an external application). All commands are case-insensitive. But some parameters may be case-sensitive.

The command `help` or `h` will show you a list of all available commands.

- `Reload` Reload the configuration.
- `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: 21  Terminals: 17  Gateways: 4  NATed: 2
Cached Endpoints: 1  Terminals: 1  Gateways: 0
-- Call Statistics --
```

```
Current Calls: 1 Active: 1 From Neighbor: 0 From Parent: 0
Total Calls: 1539  Successful: 1076  From Neighbor: 60  From Parent: 5
Startup: Fri, 21 Jun 2002 10:50:22 +0800   Running: 11 days 04:22:59
;
```

- `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>
[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>
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>
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>
Number of Endpoints: 2
;
```

- `PrintCurrentCalls`, `c`, `!` Show all current calls using the same ACF syntax as in call establishment.

  Format:

```
CurrentCalls
Call No. # | CallID | Call_Duration | Left_Time
Dialed_Number
ACF|Caller_IP:Port|Caller_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered;
ACF|Callee_IP:Port|Callee_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered;
...
Number of Calls: Current_Call Active: Active_Call From Neighbor: Call_From_Neighbor \
From Parent: Call_From_Parent
;
```

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;
ACF|10.1.1.1:1720|4037_CGK1|25263|gunter:h323_ID|frank:h323_ID|true;
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
;
```

- `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;
ACF|Callee_IP:Port|Callee_EPID|CRV|DestinationInfo|SrcInfo|IsAnswered;
# Caller_Aliases|Callee_Aliases|Bandwidth|Connected_Time <r>
...
Number of Calls: Current_Call Active: Active_Call From NB: Call_From_Neighbor
;
```

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;
ACF|10.0.1.7:1720|1325_endp|19618|gunter:h323_ID|frank:h323_ID|true;
# Sherry:h323_ID|Accel-GW1:h323_ID|200000|Wed, 26 Jun 2002 17:29:55 +0800 <2>
Number of Calls: 1 Active: 1 From NB: 0
;
```

- `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
SoftPBX: 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.

  Format:

  ```
  UnregisterIP IP[:Port]
  ```

  Example:

  ```
  UnregisterIP 10.0.1.31:1720
  URQ|10.0.1.31:1032|1326_endp|maintenance;
  SoftPBX: 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;
  SoftPBX: 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
```

- `DisconnectIP` Disconnect all calls of an endpoint by IP and call signaling port.

  Format:

  ```
  DisconnectIP IP[:Port]
  ```

  Example:

  ```
  DisconnectIP 10.0.1.31:1720
  ```

- `DisconnectAlias` Disconnect all calls of 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:

  ```
  DisconnectAlias Alias
  ```

  Example:

  ```
  DisconnectAlias 601
  ```

- `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 SEC PAR` Read and print a config parameter in a section.
  - `set SEC PAR VAL` Write a config value parameter in a section.
  - `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
  ```

- `Who` Show all people on the status port.
- `RouteReject` Terminate this call on a virtual queue. This command is used as a response to a RouteRequest event (see below).

  Format:

  ```
  RouteReject CallingEndpointID CallRef
  ```

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).

  Format:

  ```
  RouteToAlias Alias CallingEndpointID CallRef
  ```

  Example:

  ```
  RouteToAlias Suzi endp_4711 1234
  ```

- `RouteToGateway`, `rtg` Route this call on a virtual queue to the specified alias and set the destinationSignallAddress. 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.

  Format:

  ```
  RouteToGateway Alias IP:Port CallingEndpointID CallRef
  ```

  Example:

  ```
  RouteToGateway Suzi 192.168.0.50 endp_4711 1234
  ```

- `Exit`, `q` Quit the status port.

- `TransferCall` Transfer an established call from alias A to alias B. When before alias A is talking with alias X, then alias A is talking with alias B after the TransferCall.

  Currently this works only with endpoints that properly support Q.931 Facility messages (so it doesn't work with Netmeeting).

  Format:

  ```
  TransferCall Source-Alias New-Destination-Alias
  ```

  Example:

  ```
  TransferCall Frank Peter
  ```

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];` The gatekeeper receives an AdmissionRequest (ARQ) and responds with an AdmissionConfirm (ACF). The CallID is only sent when SignalCallId=1 is set.

- `ARJ|Caller_IP:Port|DestinationInfo|SrcInfo|IsAnswered|RejectReason[|CallID];` The gatekeeper receives an AdmissionRequest (ARQ) and responds with an AdmissionReject (ARJ). The CallID is only sent when SignalCallId=1 is set.
- `DCF|IP|EndpointID|CRV|DisengageReason[|CallID];` The gatekeeper receives a DisengageRequest (DRQ) and responds with a DisengageConfirm (DCF). The CallID is only sent when SignalCallId=1 is set.
- `DRJ|IP|EndpointID|CRV|RejectReason[|CallID];` The gatekeeper receives a DisengageRequest (DRQ) and responds with a DisengageReject (DRJ). The CallID is only sent when SignalCallId=1 is set.
- `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 shall respond with an InfoRequestResponse (IRR) immediately.
- `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];` Request for an external application to route an incoming call on a virtual queue. This can be done with a RouteToAlias or RouteReject command. The CallID is only sent when SignalCallId=1 is set.