OpenH323 Gatekeeper - The GNU Gatekeeper

Mainteneur de la version anglaise de ce manuel: *Jan Willamowius* <*jan@willamowius.de*>

Ceci est le Manuel Utilisateur comment compiler, installer, configurer et surveiller OpenH323 Gatekeeper - The GNU Gatekeeper.

Table des matières

1 Introduction

1.1 A propos

OpenH323 Gatekeeper - The GNU Gatekeeper <http://www.gnugk.org/> est un projet open-source qui implémente un gatekeeper H.323. Un gatekeeper fournit des services de contrôle d'appel pour les terminaux H.323. Il s'agit une partie essentielle de la plupart des installations de téléphonie sur internet qui sont basées sur la norme H.323.

Selon la recommandation H.323, un gatekeeper doit fournir les services suivants :
– Traduction d'Adresse
– Contrôle d'Admissions
– Contrôle de Bande Passante
– Gestion de Zone
– Call Control Signaling
– Autorisation d'Appel
– Gestion de Bande Passante
– Gestion des Appels
Le GNU Gatekeeper implémente la plupart des fonctions basées sur la pile du procole *OpenH323* <http://sourceforge.net/projects/openh323>

La recommandation H.323 est une norme internationale publiée par l' *ITU* <http://www.itu.int/>. Il s'agit d'une norme de communication pour l'audio, vidéo et données sur Internet. Voir aussi *l'introduction à la série de normes H.323* <http://www.packetizer.com/voip/h323/papers/primer>. de Paul Jones.

Pour une description de ce que fait un gatekeeper, voir *ici* <http://www.iec.org/online/tutorials/h323/topic06.html>.

1.2 Copyright

Il est couvert par la *GNU General Public License* (GNU GPL). En supplément de celle-ci, nous autorisons explicitement de lier ce code à la librairie OpenH323 et OpenSSL.

D'une manière générale, la GNU GPL vous autorise à copier, distribuer, revendre ou modifier les logiciels, mais elle requière que toutes les créations dérivées soient aussi publiées sous GNU GPL. Ceci signifie que vous devez publier tout le code source de toutes les extensions à gatekeeper et de tous les programmes qui incluent gatekeeper. Voir le fichier COPYING pour les détails.

Si ce n'est pas ce que vous voulez, vous devez vous interfacer au gatekeeper au travers du port d'état et communiquer par TCP avec lui. De cette façon vous devez seulement intégrer les fonctions de base dans le gatekeeper (et en fournir les sources) et conserver les autres parties privées de votre application.

1.3 Nom

Le nom exact de ce projet est *OpenH323 Gatekeeper - The GNU Gatekeeper*, en résumé *GnuGk*. Merci de ne pas le confondre avec d'autres projets de gatekeeper.

Il y a plusieurs projets gatekeeper open-source basés sur la pile de protocole OpenH323.

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

  Un gatekeeper disponible sous MPL. Le projet a été inactif pour une certaine durée.

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

  Only in a very primary grades.

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

  Celui-ci, aussi appelé GnuGk.

Avoir différents gatekeepers avec des noms similaires embrouille vraiment la plupart des utilisateurs. Comme notre "OpenH323 Gatekeeper" était le premier sur scène, ce n'est pas notre faute si d'autres ont choisi des noms similaires. Mais pour rendre la distinction plus évidente sans embrouiller encore plus les gens, nous avons décidé de donner un sous-titre au projet "OpenH323 Gatekeeper - The GNU Gatekeeper" et commencé à utiliser `gnugk` comme nom pour les exécutables.

1.4 Caractéristiques

La version 2.2.2 contient les caractéristiques et corrections suivantes :

- 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 mutiple 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.
- La valeur par défaut de la variable de configuration `ForwardOnFacility` a été changée à `0`.
- Possibilité d'encoder tous les mots de passe dans la configuration. Nouvelle variable de configuration`EncrypAllPasswords`, utilisation étendue de la variable de configuration `KeyFilled`.
- La possibilité de lire les paramètres de configuration depuis une base SQL a été portée depuis la branche 2.0. Lire 11.4 ([SQLConfig]) pour de plus amples informations.

- Framed-IP-Address ne pouvait pas être déterminé pour les appels non enregistrés sans champ, SetupUUIE.sourceCallSignalAddress, ce qui fait échouer l'authentification.
- Fournit une gestion adéquate des alias du type partyNumber (e164Number ou privateNumber).
- Une correction pour RTP/Q931/H245/T120PortRange pour corriger une anomalie avec le bouclage du domaine des ports si le dernier port est 65535. Ceci amenait le port suivant à être mis à 0 et les allocation ultérieures de port échouaient.
- L'allocation dynamique de ports RTP ne marchait pas, utilise un domaine de ports figé 1024-65535 comme valeur par dé pour la variable de configuration `RTPPortRange`.
- Les modules auth obsolètes MySQLAliasAuth et MySQLPasswordAuth sont supprimés.
- Les modules SQL acceptent un seul serveur de base de données dans le paramètre `Host`.

La version 2.2.1 contient les caractéristiques et corrections suivantes :

- Amélioration de la correspondance des préfixes pour les politiques de routage. Un point (.) correspond à n'importe quel chiffre.
- Amélioration de la correspondance des voisins. Un point (.) correspond à n'importe quel chiffre. `!` au début désactive le préfixe.
- Un verrou manquant pendant le rechargement de la configuration faisait crasher le gatekeeper.
- Sélection plus fiable de numéro de port pour les plages de port de Q.931, H.245, T.120 et RTP (avant, un rechargement de la configuration pouvait cause des erreurs pour beaucoup d'appels à, cause de l'impossibilité d'allouer une nouvelle socket).
- La valeur par défaut de RTPPortRange est maintenant de laisser l'OS sélectionner un numéro de port.
- Règles de réécriture plus flexibles (global et par passerelle) avec les nouveaux caractères wildcard '.' et '%'.
- Amélioration de la correspondance des préfixes pour les passerelles. Un point (.) correspond à n'importe quel chiffre. `!` au début désactive le préfixe.
- Insère le Calling-Party-Number-IE/Display-IE manquant si les otpions correspondantes `Screen...` sont activées.
- Arrête le gatekeeper si il y a des erreurs dans la configurationd es modules SQL aut/acct
- Le type de numéro Called-Station-Id peut être sélectionné entre le numéro original (numéro composé) et le numéro réécrit. Nouvelle option de configurgation `UseDialedNumber` pour les modules 7.7 (RadAuth)/7.8 (RadAliasAuth) /8.3 (RadAcct), nouvelle variable %{Dialed-Number} pour les modules 8.4 (SQLAcct) et 8.2 (FileAcct).
- Possibilité de modifier les formats d'horodate. Nouvelle variable de configuration `TimestampFormat` pour les parties main, 8.4 ([SqlAcct]), 8.3 ([RadAcct]), 8.2 ([FileAcct]) et 11.1 ([CallTable]).
- Les modules RadAuth/RadAliasAuth peuvent maintenant ajouter/supprimer des alias de terminaux pendant l'enregistrement de terminaux (en utilisant h323-ivr-in=terminal-alias : Cisco AV-Pair).
- Nouvelle option `TcpKeepAlive` pour régler le problème avec les erreurs réseau et les appels bloqués dans la table d'appel. Voir docs/keepalive.txt pour de plus amples informations.
- Nouvelle commande du port d'état `RouteToGateway`.

La version 2.2.0 contient les caractéristiques et corrections suivantes :

- Nouvelle option de configuration `RoundRobinGateways`.
- Limites de capacité d'appel et routage des priorités pour les passerelles. Nouvelles sections de configuration `EP : :` pour des paramètres de configuration par terminal (voir 10 (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 separatelly.
- New, much faster, Radius client implementation.
- Called-Party-Number-IE rewrite occured 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 (8.4 ([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 signalling 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 tunnelling 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. MySQL and LDAP are supported as backend database for authentication.
- 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 resigtering 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 Téléchargement

La dernière version stable et une version de développement sont disponibles sur *la page de téléchargement* <http://www.gnugk.org/h323download.html>.

La toute dernière version du code source est sous CVS sur *Sourceforge* <http://sourceforge.net/cvs/?group_id=4797> (*Web-GUI* <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/openh323gk/>). Attention - il s'agit de la pointe de la technologie.

Vous pouvez aussi télécharger certains exécutables depuis *la page de téléchargement* <http://www.gnugk.org/h323download.html>.

1.6 Liste de Diffusion

Il y a deux listes de diffusion pour le projet, une pour les développeurs et une pour les utilisateurs.

Les questions d'ordre général doivent être envoyées à la *liste de diffusion des utilisateurs* <mailto:Openh323gk-users@sourceforge.net>. Vous pouvez trouver les archives de la liste *ici* <http://sourceforge.net/mailarchive/forum.php?forum_id=8549>. Pour vous joindre à cette liste de diffusion, cliquez *ici* <http://lists.sourceforge.net/lists/listinfo/openh323gk-users>.

Pour signaler des problèmes ou des anomalies/patches, envoyer des mails à la *liste de diffusion des développeurs* <mailto:Openh323gk-developer@sourceforge.net>. Les archives de la liste sont *ici* <http://sourceforge.net/mailarchive/forum.php?forum_id=3079>. Veuillez envoyer les questions utilisateur à la liste de diffusion des utilisateurs et garder cette liste pour le développement ! Si vous voulez contribuer au projet, veuillez *rejoindre la liste de diffusion* <http://lists.sourceforge.net/lists/listinfo/openh323gk-developer>.

Note : Merci de ne pas envoyer vos questions par des mails privés aux développeurs. Nous sommes habituellement occupés. Nous ne voulons **pas** être votre consultant personnel, à moins que vous souhaitiez nous payer. Envoyez vos problèmes à la liste de diffusion appropriée de telle sorte que tout le monde puisse vous aider.

Merci aussi de ne pas envoyer les problèmes spécifiques de GnuGk à la liste de diffusion OpenH323, ou vice versa. Il s'agit de projet distincts, bien que très apparentés.

Avant d'envoyer un mail, assurez-vous d'avoir lu les documents associés méticuleusement. Décrivez votre problème clairement et avec précision. Montrez nous les messages d'erreur ou les logs si il y en a.

1.7 Participants

Le coordinateur actuel du projet est *Jan Willamowius* <http://www.willamowius.de/> <*jan@willamowius.de*>

Les principales caractéristiques et fonctions de la version 2.0 ont été contribuées par *Chih-Wei Huang* <http://www.cwhuang.idv.tw/> <*cwhuang@linux.org.tw*> et *Citron Network Inc.* <http://www.citron.com.

tw/>, incluant l'enregistrement et les tables d'appel thread safe, une nouvelle architecture de mode routé, proxy H.323, authentification H.235 et MySQL backend.

Michal Zygmuntowicz <*m.zygmuntowicz@onet.pl*> a fait du bon travail sur le support de Radius et d'autres améliorations.

La version initiale de gatekeeper a été développée par Xiang Ping Chen, Joe Metzger et Rajat Todi.

2 Compilation et Installation

2.1 Compilation du Gatekeeper

Pour construire le gatekeeper vous avez besoin d'au moins PWLib 1.5.0 et OpenH323 1.12.0 ou ultérieur. La version de développement du gatekeeper a généralement besoin de la version de OpenH323 la plus récente disponible. Ces librairies sont disponibles sur la *Page de Téléchargement de OpenH323* <http://sourceforge.net/projects/openh323>. Voir les instructions sur *comment compiler le code de OpenH323* <http://www.openh323.org/build.html>.

Ordre de compilation :

1. PWLib (version release + debug)
2. OpenH323
3. Application de test OpenH323 (pas nécessaire, juste pour s'assurer que tout marche)
4. Le Gatekeeper

Sous Unix faire un `configure` et `make debug` ou `make opt` dans le répertoire du gatekeeper pour construire la version debug ou release, respectivement. Utiliser `make both` pour construire les deux versions. Il faut noter que vous devez utiliser GCC 3.3.x ou ultérieur. Les versions plus anciennes peuvent ne pas marcher. Une bonne habitude est de faire un `make debugdepend` ou `make optdepend` dans le répertoire du gatekeeper avant de commencer la compilation (make debug ou make opt) - ces commandes construisent les listes de dépendances appropriées, de telle sorte qu'après avoir mis à jour vos sources depuis CVS, tous les fichiers affectés seront recompilés. Sinon vous pourriez obtenir un Gatekeeper partiellement compilé avec les vieux en-têtes et partiellement avec les nouveaux en-têtes - une mauvaise chose.

Sous Windows, ouvrez et compilez simplement la solution fournie (`gk.sln`) pour Microsoft Visual Studio .NET 2003 ou le workspace (`gk.dsw`) pour Microsoft Visual Studio 6.0 SP6. Bien sur, vous devez déjà avoir compilé PWLib et OpenH323 et configuré les chemins appropriés pour les include/librairies. Si vous souhaitez le support MySQL ou PostgreSQL, installez/compilez les librairies clients appropriées et ajoutez HAS_MSQL=1 et/ou HAS_PGSQL=1 dans les macros du préprocesseur du projet gatekeeper. Vous devez aussi indiquer au compilateur ou trouver les fichiers include/librairies et indiquer à l'éditeur de liens de linker avec ces librairies clients.

Taper `configure -help` pour voir une liste détaillée de toutes les options de compilation. Vous pouvez les utiliser pour activer ou désactiver certaines fonctions du gatekeeper. Par exemple, si vous n'avez pas besoin de RADIUS, taper juste : `configure -disable-radius`.

Les versions recommandées de PWLib/OpenH323 sont celles de la version Pandora (1.7.5.2/1.14.4.2) ou plus récente. Les versions plus anciennes ne sont plus supportées et ne sont pas garanties de fonctionner avec le gatekeeper.

Pour construire le gatekeeper qui est lié statiquement avec les librairies système et OpenH323, il faut utiliser `make optnoshared` ou `make debugnoshared`.

Pour utiliser le gatekeeper sous une charge lourde, activer la fonction LARGE_FDSET (SEULEMENT POUR LA VERSION UNIX) est recommandé (configure –with-large-fdset=4096). Certains systèmes requièrent aussi l'utilisation de ulimit pour autoriser l'allocation de plus de 1024 sockets pour un seul processus. Il faut noter qu'à partir de la version 1.5.3 la librairie PWLib support aussi une fonction similaire, vous pouvez donc choisir entre les implémentations LARGE_FDSET GnuGk et PWLib. L'implémentation native GnuGk donne de meilleures performances. La valeur maximale LARGE_FDSET devrait êtra calculée en se basant sur l'utilisation maximum prévue de sockets. Un procédé empirique est :

```
MAX_NUMBER_OF_CONCURRENT_CALLS * 10 * 120%

Où:
10 = 2 sockets pour Q.931 + 2 sockets pour H.245 + 6 sockets pour RTP et autres
```

Ainsi pour 100 appels simultanés vous n'avez pas besoin de plus d'environ 1024 sockets dans le LARGE_FDSET.

2.2 L'utilitaire addpasswd

L'authentification de l'accès à la ligne d'état et le module SimplePasswordAuth ont besoin que des mots de passe cryptés soient stockés dans le fichier de configuration du gatekeeper. Et depuis la version 2.2.2, le gatekeeper supporte le cryptage de tous les mots de passe dans la configuration. L'utilitaire `addpasswd` est nécessaire pour générer et stocker les mots de passe cryptés. Cet utilitaire est fourni avec le gatekeeper et peut être compilé en utilisant :

```
$ make addpasswd
```

L'utilisation est la suivante :

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

Exemple 1 : l'utilisateur 'gkadmin' avec le mot de passe 'secret' doit être ajouté à la section de configuration [GkStatus : :Auth] pour activer l'interface de la ligne d'état :

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

Exemple 2 : l'utilisateur 'joe' avec le mot de passe 'secret' doit être ajouté à la section de configuration [Password] pour activer l'authentification de terminal :

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

Exemple 3 : Un secret partagé crypté est ajouté à la section de configuration RadAuth :

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

IMPORTANT : La variable `KeyFilled` définit une initialisation par défaut pour les clés de cryptage de mot de passe. Il peut être omis dans le fichier de configuration (il est alors égal à 0), mais si il est précisé, à chaque fois qu'il change, les mots de passe cryptés doivent être regénérés (encore cryptés avec l'utilitaire `addpasswd`).

2.3 Installation du Gatekeeper

Il n'y a pas de procédure d'installation spéciale. Copiez juste l'exécutable dans le répertoire que vous voulez et créez le fichier de configuration. Il y a plusieurs exemples de configuration dans le répertoire `etc/` de l'arborescence des sources. Se reporter à la section 4.2 (Fichier de Configuration) pour de plus amples explications.

Par exemple, sur une plateforme Linux x86, l'exécutable optimisé `gnugk` est créé dans le sous-répertoire `obj_linux_x86_r/`. Vous pouvez le copier dans `/usr/sbin/`, créer une configuration dans `/etc/gnugk.ini` et le démarrer avec

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

Se reporter à la section 4.1 (Options de la Ligne de Commande) pour les détails.

2.4 Binaires Préconstruits

Si vous ne voulez pas compiler le gatekeeper à partir des sources, il y a plusieurs 'packages' préconstruits disponibles sur *SourceForge* <http://sourceforge.net/project/showfiles.php?group_id=4797>. Toutes les versions ne sont pas disponibles - vérifier ce qui est disponible.

Packages tar (.tgz ou .tar.gz)

Télécharger le fichier tar et saisir la commande suivante en tant que `root`, en substituant le nom du fichier que vous avez téléchargé.

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

Packages Debian (.deb)

Debian inclut le gatekeeper en tant que package openh323gk. Vous pouvez l'installer en utilisation la commande suivante en tant que `root` :

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

3 Pour Commencer (Tutorial)

3.1 Une première expérience simple

Pour s'assurer que tous les composants fonctionnent, trouvez 2 stations de travail Linux, toutes les deux connectées au réseau local. Assurez-vous d'avoir au moins la version 1.1 de OpenH323 et OhPhone d'installée. Sur la première machine, lancez le gatekeeper et ohphone (dans des consoles différentes) :

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

Le gatekeeper tourne maintenant en mode direct. L'option "`-ttt`" indique au gatekeeper de faire beaucoup de traces de debug dans la console (vous pouvez rediriger cette sortie vers un fichier avec "`-o logfile`").

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

Cet OhPhone attend maintenant (`-l`) les appels et les acceptera automatiquement (`-a`). Il s'est enregistré comme utilisateur jan auprès du gatekeeper qu'il détectera automatiquement. (Si la détection automatique ne marche pas pour une quelconque raison, utilisez "`-g 1.2.3.4`" pour indiquer le numéro IP sur lequel le gatekeeper tourne.)

Sur la deuxième machine, lancez seulement ohphone :

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

La deuxième instance de OhPhone s'enregistre auprès du gatekeeper détecté automatiquement en tant qu'utilisateur peter et essaye d'appeler jan. Le gatekeeper déterminera le numéro IP où jan s'est enregistré (machine1 dans ce cas) à partir du username et OhPhone appellera l'autre instance de OhPhone sur la machine une.

La première instance de OhPhone acceptera cet appel, Peter et Jan pourront parler.

3.2 Utilisation de l'interface Status pour surveiller le gatekeeper

Nous allons maintenant essayer de voir quels messages sont traités par le gatekeeper. Dans une nouvelle console sur machine1 nous utilisons telnet pour nous connecter au gatekeeper :

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

Très probablement nous aurons un message "Accès interdit !", car tout le monde n'a pas le droit d'espionner.

Nous créons maintenant un fichier appelé `gatekeeper.ini` et le mettons dans le répertoire où nous démarrons le gatekeeper. `gatekeeper.ini` contient seulement 4 lignes :

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

Arrêtez le gatekeeper avec Ctrl-C et redémarrez-le. Quand nous faisons à nouveau le telnet, nous restons connectés au gatekeeper. Répétons maintenant la première expérience où Peter appelle Jan et voyons quels messages sont traités par le gatekeeper en mode non-routé. Il existe un certain nombre de commandes qui peuvent être envoyées dans cette session telnet : Tapez "help" pour les voir. Pour terminer la session telnet avec le gatekeeper, tapez "quit" et Entrée.

3.3 Démarrage du gatekeeper en mode routé

Démarrer le gatekeeper en mode routé signifie que le gatekeeper utilise la "signalisation routée du gatekeeper" pour tous les appels. Dans ce mode tous les messages de signalisation du gatekeeper passent par le gatekeeper qui a beaucoup plus de contrôle sur les appels.

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

Le gatekeeper tourne maintenant en mode routé. Faire un telnet sur le port d'état et faîtes un appel pour voir quels messages sont maintenant traités par le gatekeeper.

Il faut noter que tous les paquets media (audio et vidéo) sont toujours envoyés directement entre les terminaux (les 2 instances de ohphone).

Comme la signalisation routée du gatekeeper est beaucoup plus complexe, vous avez plus de chances de tomber sur une anomalie dans ce mode. Mais si çà casse, vous gardez les pièces. ;-)

3.4 Un PBX virtuel : Déconnection des appels

Jusqu'à maintenant le gatekeeper a seulement servi de mécanisme pour résoudre des noms symboliques en adresses IP. Il s'agit d'une fonction importante mais pas très excitante.

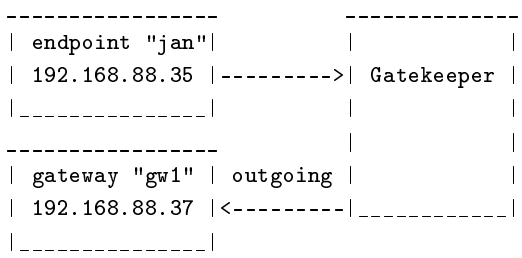
Puisque le gatekeeper a beaucoup de contrôle sur les appels, il peut les terminer par exemple. Quand nous sommes connectés au port d'état, nous pouvons obtenir la liste de tous les appels en cours avec "`PrintCurrentCalls`". Pour terminer un appel, nous pouvons dire "`Disconnectip 1.2.3.4`" pour un de ses terminaux.

Quelqu'un pourrait par exemple écrire un script simple qui se connecte au port d'état, surveille les appels en cours et les termine après 5 minutes, de telle sorte qu'aucun utilisateur ne puisse abuser des ressources système.

Regardez les autres fonctions téléphoniques telles que TransferCall pour voir ce qui est disponible.

3.5 Routage d'appels sur une passerelle pour atteindre des utilisateurs externes

Sans utiliser de passerelle vous pouvez seulement appeler d'autres personnes avec un téléphone IP sur Internet. Pour atteindre les gens disposant d'un téléphone ordinaire vous devez utiliser une passerelle.



Le gatekeeper doit savoir quels appels sont supposés être routés par la passerelle et quels numéros doivent être appelés directement. Utilisez la section [RasSrv : :GWPrefixes] du fichier de configuration pour indiquer au gatekeeper le préfixe des numéros qui doivent être routés par la passerelle.

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

Cette entrée indique au gatekeeper de router tous les appels aux numéros E.164 qui commencent par 0 sur la passerelle qui s'est enregistrée avec l'alias H.323 "gw1". Si il n'y a pas de passerelle enregistrée avec cet alias l'appel échouera. (Il faut noter que vous devez utiliser l'alias de la passerelle - vous ne pouvez pas juste indiquer au gatekeeper le numéro IP de la passerelle.)

Un préfixe peut contenir des chiffres `0-9`, `#` et `*`. Il peut aussi contenir un carctère spécial `.` (un point) qui correspond à n'importe quel chiffre et peut être préfixé par `!` (un point d'exclamation) pour désactiver le préfixe. La correspondance des préfixes est faite en respectant la règle du plus long préfixe correspondant, avec les règles! ayant une priorité supérieure si les longueurs sont égales. Quelques exemples :

```
[RasSrv::GWPrefixes]
; Cette entrée routera les numéros commençant par 0048 (mais pas par
; 004850 et 004860) vers gw1
gw1=0048,!004850,!004860
; Cette entrée correspond uniquement à 001 avec 10 chiffres après
gw2=001.........
```

3.6 Réécriture de numéros E.164

Quand vous utilisez une passerelle vous devez souvent utiliser des numéros différents en interne et les réécrire avant de les envoyer par la passerelle sur le réseau téléphonique. Vous pouvez utiliser la section 5.2 (RasSrv : :RewriteE164) pour configurer ceci.

Exemple : Vous voulez appeler le numéro 12345 avec votre téléphone IP et atteindre le numéro 08765 derrière la passerelle "gw1".

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

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

Vous pouvez aussi configurer la réécriture de numéros E.164 en fonction de quelle passerelle vous recevez un appel ou vous en envoyez un en utilisant la section 5.3 (RasSrv : :GWRewriteE164).

Exemple : Vous avez deux passerelles différentes ("gw1" et "gw2") auxquelles vous envoyez des appels avec le préfixe 0044, mais qui nécessitent l'ajout d'un préfixe différent après que le routage ait choisi la passerelle. Ce peut être par exemple pour des raisons d'identification.

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

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

Exemple : Vous voulez identifier les appels d'une passerelle particulière "gw1" avec un préfixe spécifique avant de passer ces appels à une autre passerelle "gw2".

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

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

Les expressions de réécriture acceptent les caractères jokers point '.' et pourcent '%' pour permettre de construire des règles plus générales. Le caractère point peut apparaître à la fois à gauche et à droite de l'expression, le signe pourcent peut apparaître uniquement à la gauche de l'expression. Utilisez '.' pour indiquer n'importe quel caractère et le copier dans la chaîne réécrite et '%' pour indiquer n'importe quel caractère et l'omettre. Quelques exemples simples :

```
[RasSrv::RewriteE164]
; Réécrit 0044 + min. 7 chiffres en 44 + min. 7 digits
0044.......=44.......
; Réécrit les numéros commencant par 11 + 4 chiffres + 11 en 22 + 4 digits + 22
; (comme 11333311 => 22333322, 110000112345 => 220000222345)
11....11=22....22
; omet les 4 premiers chiffres de tous les numéros (11114858345 => 4858345)
; c'est équivalent à 10 règles %%%%1=1, %%%%2=2, ...
%%%%.=.
; insère deux zéros au milieu du numéro (111148581234 => 11110048581234)
....48=....0048
; même ceci est possible (415161 => 041051061)
4.5.6=04.05.06
```

4 Configuration Basique du Gatekeeper

Le comportement du gatekeeper est complètement déterminé par les options de la ligne de commande et le fichier de configuration. Certaines options de la ligne de commande peuvent annuler certains paramètres du fichier de configuration. Par exemple, l'option `-l` annule le paramètre `TimeToLive` dans le fichier de configuration.

4.1 Options de la Ligne de Commande

Presque chaque option a un format court et un format long, i.e., `-c` est comme `-config`.

4.1.1 Basique

`-h -help`

Liste toutes les options disponibles et quitte le programme.

`-c -config filename`

Indique le fichier de configuration à utiliser.

`-s -section section`

Indique quelle section principale utiliser dans le fichier de configuration. Par défaut, c'est [Gatekeeper : :Main].

`-i -interface IP`

Indique l'interface (numéro IP) sur laquelle écoute le gatekeeper. Vous devriez omettre cette option pour laisser le gatekeeper déterminer automatiquement l'IP sur laquelle il écoute, à moins que vous ne vouliez que le gatekeeper s'attache à une IP précise.

`-l -timetolive n`

Indique la minuterie (en secondes) du temps-à-vivre pour l'enregistrement de terminal. Il annule le paramètre `TimeToLive` du fichier de configuration. Voir 4.2.1 (ici) pour des explications détaillées.

`-b -bandwidth n`

Indique la bande passante totale disponible pour le gatekeeper. En ne précisant pas cette option, la gestion de la bande passante est désactivée par défaut.

`-pid filename`

Indique le fichier pid, valable uniquement pour les versions Unix.

`-u -user name`

Exécute le processus gatekeeper avec cet utilisateur. Valable uniquement pour les version Unix.

`-core n`

(Unix seulement) Permet l'écriture de fichiers core dump quand l'application plante. Un fichier core dump ne dépassera pas la taille de n octets. Une constante spéciale "unlimited" peut être utilisée pour ne pas imposer de limite.

4.1.2 Mode du Gatekeeper

Les options de cette sous-section annulent les paramètres de la 4.2.4 (section [RoutedMode]) du fichier de configuration.

`-d -direct`

Utilise le signal d'appel direct de terminal.

`-r -routed`

Utilise le signal d'appel routé du gatekeeper.

`-rr -h245routed`

Utilise le signal d'appel routé du gatekeeper et le canal de contrôle H.245.

4.1.3 Information de Debug

`-o -output filename`

Ecrit les traces dans le fichier indiqué.

`-t -trace`

Règle le niveau de trace. Plus vous ajoutez de `-t`, plus les traces sont complètes. Par exemple, utiliser `-ttttt` pour régler le niveau de trace à 5.

4.2 Fichier de Configuration

Le fichier de configuration est un fichier texte normal. Le format de base est :

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

Les commentaires sont marqués avec un dièse (#) ou un point virgule (;) au début de la ligne.

Le fichier `complete.ini` contient toutes les sections disponibles pour GnuGk. Dans la plupart des cas, il n'est pas nécessaire de tous les utiliser. Ce fichier est juste une collection d'exemples pour beaucoup de réglages.

Le fichier de configuration peut être modifié pendant l'exécution. Une fois que le fichier de configuration est modifié, vous pouvez envoyer une command `reload` par le port d'état, ou envoyer un signal `HUP` au processus du gatekeeper sous Unix. Par exemple,

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

4.2.1 Section [Gatekeeper : :Main]

- `Fortytwo=42`
  Défaut : `N/A` Ce paramètre est utilisé pour tester la présence du fichier de configuration. Si il n'est pas trouvé, un avertissement est émis. Assurez-vous qu'il soit dans tous vos fichiers de configuration.
- `Name=OpenH323GK`
  Défaut : `OpenH323GK` Identifiant Gatekeeper de ce gatekeeper. Le gatekeeper répondra uniquement aux GRQs pour cet ID et l'utilisera dans un certain nombre de messages à ces terminaux.
- `Home=192.168.1.1`
  Défaut : `0.0.0.0` Le gatekeeper écouter les requêtes sur ce numéro IP. Par défaut, le gatekeeper écoute toutes les interfaces de votre hote. Vous devriez renoncer à cette option, à moins que vous ne souhaitiez que le gatekeeper ne s'attache qu'à une IP spécifique. Des adresses multiples Home peuvent être utilisées et doivent être séparéres par un point-virgule (;) ou une virgule (,).
- `NetworkInterfaces=192.168.1.1/24,10.0.0.1/0`
  Défaut : `N/A` Indique les interfaces réseau du gatekeeper. Par défaut le gatekeeper détectera les interfaces de vote hote automatiquement. Il y a deux situations où vous voudriez utiliser cette option. La première est que la détection automatique a échoué, l'autre est que le gatekeeper soit derrière un routeur et autorise les terminaux avec des IPs publiques à s'enregistrer.
- `EndpointIDSuffix=_gk1`
  Défaut : `_endp` Le gatekeeper assignera un identifiant unique à chaque terminal enregistré. Cette option peut être utilisée pour préciser un suffixe à ajouter à l'identifiant du terminal. Ceci est seulement utile quand vous utilisez plus d'un gatekeeper.
- `TimeToLive=300`
  Défaut : `-1` L'enregistrement d'un terminal avec un gatekeeper peut avoir une durée de vie limitée. Le gatekeeper précise la durée de l'enregistrement d'un terminal en incluant un champ **timeToLive** dans le message RCF. Après le temps indiqué, l'enregistrement a expiré. Le terminal devrait périodiquement envoyer un RRQ ayant le bit **keepAlive** positionné avant l'expiration. Un tel message peut inclure une quantité minimum d'informations comme décrit dans H.225.0. Ceci est appelé un RRQ léger.
  Ce paramètre de configuration indique la minuterie en secondes du temps à vivre avant que l'enregistrement se termine. Il faut noter que le terminal peut demander un **timeToLive** plus court dans le message RRQ au gatekeeper. Pour éviter une surcharge de messages RRQ, le gatekeeper ajuste automatiquement cette minuterie à 60 secondes si vous indiquez une valeur inférieure !
  Après l'expiration du délai, le gatekeeper enverra par la suite deux messages IRQ pour demander si le terminal est encore vivant. Si le terminal répond avec un IRR, l'enregistrement sera étendu. Autrement le gatekeeper enverra un URQ avec la raison **ttlExpired** au terminal. Le terminal doit alors se ré-enregistrer auprès du gatekeeper en utilisant un message RRQ complet.

Pour désactiver cette fonction, lui donner la valeur `-1`.

- `TotalBandwidth=100000`
  Défaut : `-1` La bande passante totale disponible à donner aux terminaux. Par défaut cette fonction est désactivée. Faîtes attention quand vous l'utilisez car beaucoup de terminaux ont des implémentations erronées.
- `RedirectGK=Endpoints > 100 || Calls > 50`
  Défaut : `N/A` Cette option vous permet de rediriger des terminaux vers des gatekeepers alternatifs quand le gatekeeper est surchargé. Par exemple, avec le paramètre ci-dessus le gatekeeper rejettera un RRQ si le nombre de terminaux enregistrés excède 100, ou rejettera un ARQ si les appels concurrents excèdent 50. De plus, vous pouvez explicitement rediriger tous les terminaux en positionnant cette option à `temporary` ou `permanent`. Le gatekeeper retournera un message de rejet RAS avec une liste de gatekeepers alternatifs définis dans `AlternateGKs`. Il faut noter que la redirection `permanent` signifie que les terminaux redirigés ne s'enregistreront plus avec ce gatekeeper. Veuillez aussi noter que la fonction entre en vigueur avec les terminaux respectant la version 4 de H.323.
- `AlternateGKs=1.2.3.4 :1719 :false :120 :OpenH323GK`
  Défaut : `N/A` Nous autorisons l'existence d'un autre gatekeeper pour fournir de la redondance. Ceci est implémenté de manière active-active. En réalité, vous pouvez vous retrouver dans une situation (valide!) où certains terminaux sont enregistrés avec le premier gatekeeper et d'autres sont enregistrés avec le second. Vous pourriez même utiliser les deux gatekeepers d'une manière round_robin(?) pour le partage de charge (ce n'est pas testé :-)). Si vous continuez à lire, "GK primaire" indique le gatekeeper que vous êtes en train de configurer and "GK alternatif" indique l'autre. The GK primaire inclut un champ dans le RCF pour indiquer aux terminaux quels IP et gatekeeper alternatifs utiliser. Mais le GK alternatif doit connaître chaque enregistrement avec le GK primaire ou il rejetterait les appels. Donc notre gatekeeper peut transmettre chaque RRQ à l'adresse IP alternative.
  L'option de configuration AlternateGKs indique les champs contenus dans le RCF du GK primaire. Le premier et le deuxième champ de cette chaîne indique où transmettre (IP, port). Le troisième indique aux terminaux si ils doivent s'enregistrer auprès du GK alternatif avant de passer des appels. Il n'ont généralement pas besoin car nous transmettons leurs RRQs, ils sont donc enregistrés auprès du GK alternatif. Le quatrième champ indique la priorité pour ce GK. Plus bas est mieux, en géné ral on considère que le GK primaire a une priorité de 1. Le dernier champ indique l'identifiant du gatekeeper alternatif.
- `SendTo=1.2.3.4 :1719`
  Défaut : `N/A` Bien que cette information soit contenue dans AlternateGKs, vous devez quand même indiquer à quelle adresse transmettre les RRQs. Ceci peut être différent de l'adresse de AlternateGK, il s'agit donc d'une option de configuration séparée (penser aux machines à plusieurs adresses).
- `SkipForwards=1.2.3.4,5.6.7.8`
  Défaut : `N/A` Pour éviter les transmissions circulaires, vous ne devez pas transmettres les RRQs que vous recevez de l'autre GK (cette déclaration est valable pour les deux, GK primaire et alternatif). Deux mécanismes sont utilisés pour identifier si une requête doit être transmise. Le premier cherche un drapeau dans le RRQ. Comme peu de terminaux implémentent ceci, nous avons besoin d'une deuxième façon plus fiable. Indiquez l'IP de l'autre gatekeeper dans cette liste.
- `StatusPort=7000`
  Défaut : `7000` Port d'état pour surveiller le gatekeeper. Se reporter à 12 (cette section) pour plus de détails.
- `SignalCallId=1`
  Défaut : `0` IDs du signal d'appel dans les messages ACF/ARJ/DCF/DRJ/RouteRequest sur le port d'état. Se reporter ô 12 (cette section) pour plus de détails.
- `StatusTraceLevel=2`
  Défaut : `2` Niveau de trace par défaut pour les clients de la nouvelle interface d'état. Se reporter à 12 (cette section) pour plus de détails.
- `TimestampFormat=ISO8601`
  Défaut : `Cisco` Contrôle le format par défaut des chaînes horodate générées par le gatekeeper. Cette

option affecte 8.4 ([SqlAcct]), 8.3 ([RadAcct]), 8.2 ([FileAcct]) et d'autres modules, sauf 11.1 ([CallTable]). Vous pouvez adapter encore plus le formatage des horodates par module en configurant le paramètre `TimestampFormat` par module.
Il y a quatre formats prédéfinis :
- `RFC822` - un format par défaut utilisé par le gatekeeper (exemple : Wed, 10 Nov 2004 16 :02 :01 +0100)
- `ISO8601` - format standard ISO (exemple : 2004-11-10 T 16 :02 :01 +0100)
- `Cisco` - format utilisé par un équipement Cisco (exemple : 16 :02 :01.534 CET Wed Nov 10 2004)
- `MySQL` - format simple que MySQL peut comprendre (exemple : 2004-11-10 16 :02 :01)

Si vous avez besoin d'un autre format, vous pouvez construire votre propre chaîne de format, en utilisant des règles tirées de la fonction C `strftime` (voir man strftime ou chercher sur MSDN pour strftime). En général, la chaîne de format se compose de caractères normaux et de codes de formatage, précédés d'un signe pourcent. Exemple : "%Y-%m-%d et pourcent %%" donnera comme résultat "2004-11-10 et pourcent %". Quelques codes de format habituels :
- `%a` - nom abrégé du jour de la semaine
- `%A` - nom complet du jour de la semaine
- `%b` - nom abrégé du mois
- `%B` - nom complet du mois
- `%d` - jour du mois sous la forme d'un nombre
- `%H` - heure en format 24-heures
- `%I` - heure en format 12-heures
- `%m` - mois sous la forme d'un nombre
- `%M` - minute sous la forme d'un nombre
- `%S` - seconde sous la forme d'un nombre
- `%y` - année sans le siècle
- `%Y` - année avec le siècle
- `%u` - microsecondes sous la forme d'un nombre (**extension GnuGk**)
- `%z` - abréviation du fuseau horaire (+0100)
- `%Z` - nom du fuseau horaire
- `%%` - signe pourcent

- `EncryptAllPasswords=1`
  Défaut : `0` Active le chiffrage de tous les mots de passe dans la configuration (mots de passe SQL, RADIUS, [Password], [GkStatus : :Auth]). Si c'est activé tous les mots de passe doivent être cryptés en utilisant l'utilitaire `addpasswd`. Autrement seuls les mots de passe de [Password] et [GkStatus : :Auth] sont cryptés (ancien comportement).
- `KeyFilled=0`
  Défaut : `N/A` Définit un octet de remplissage global à utiliser pendant le cryptage/ décryptage de mot de passe. Il peut être annulé en positionnant `KeyFilled` dans une section de configuration particulière. Habituellement, vous n'avez pas besoin de changer cette option.

La plupart des utilisateurs n'auront jamais besoin de modifier une des valeurs suivantes. Ils sont principalement utilisés pour les tests ou pour des applications trè sophistiquées.

- `UseBroadcastListener=0`
  Défaut : `1` Indique si on doit écouter au requêtes RAS diffusées. Ceci implique de s'attacher à toutes les interfaces sur la machine, donc si vous voulez lancer plusieurs instances de gatekeepers sur la même machine, vous devriez désactiver cette option.
- `UnicastRasPort=1719`
  Défaut : `1719` L'identifiant TSAP du canal RAS pour unicast(?).
- `MulticastPort=1718`
  Défaut : `1718` L'identifiant TSAP du canal RAS pour multicast.
- `MulticastGroup=224.0.1.41`
  Défaut : `224.0.1.41` Le groupe de multicast pour le canal RAS.

- `EndpointSignalPort=1720`
  Défaut : `1720` Default port for call signalling channel of endpoints.
- `ListenQueueLength=1024`
  Défaut : `1024` Longueur de la queue pour les connections entrantes TCP.
- `SignalReadTimeout=1000`
  Défaut : `1000` Temps en millisecondes pour le timeout de lecture sur les canaux de signal d'appel (Q931).
- `StatusReadTimeout=3000`
  Défaut : `3000` Temps en millisecondes pour le timeout de lecture sur le canal d'état.
- `StatusWriteTimeout=5000`
  Défaut : `5000` Temps en millisecondes pour le timeout d'écriture sur le canal d'état.

4.2.2 Section [GkStatus : :Auth]

Définit un nombre de règles pour indiquer qui est autorisé à se connecter au port d'état. Chacun ayant accès au port d'état a le contrôle total sur votre gatekeeper. Assurez vous de le définir correctement.

- `rule=allow`
  Défaut : `forbid` Les valeurs possibles sont
  - `forbid` - interdit toute connexion.
  - `allow` - autorise toute connexion.
  - `explicit` - lit le paramètre `ip=value` où `ip` est l'adresse IP i du client observateur, `value` est `1,0` ou `allow,forbid` ou `yes,no`. Si `ip` n'est pas listée le paramètre `default` est utilisé.
  - `regex` - l'IP du client est comparée à l'expression régulière.

    Exemple :

    Pour autoriser les clients de 195.71.129.0/24 et 195.71.131.0/24 :

    ```
    regex=^195\.71\.(129|131)\.[0-9]+$
    ```

  - `password` - l'utilisateur doit saisir le nom d'utilisateur et le mot de passe appropriés pour se connecter. Le format. Le format de username/password est le même que pour la section 7.2 ([SimplePasswordAuth]).

  De plus, ces règles peuvent être combinées avec "|" ou "&". Par exemple,
  - `rule=explicit | regex`
    L'IP du client doit correspondre à la règle `explicit` **ou** `regex`.
  - `rule=regex & password`
    L'IP du client doit correspondre à la règle `regex`, **et** l'utilisateur doit saisir son nom d'utilisateur et son mot de passe.
- `default=allow`
  Défaut : `forbid` Utilisé uniquement quand `rule=explicit`.
- `Shutdown=forbid`
  Défaut : `allow` Indique si on peut arrêter le gatekeeper par le port d'état.
- `DelayReject=5`
  Défaut : `0` Combien de temps (en secondes) avant de rejeter un usernam/password invalide pour l'accès à la ligne d'état.

4.2.3 Section [LogFile]

Cette section définit les paramètres liés aux fichiers de log. Actuellement, ceci permet aux utilisateurs de préciser les options de rotation des fichiers de log.

- `Rotate=Hourly | Daily | Weekly | Monthly`
  Défaut : `N/A` Si précisé, le fichier de log tournera en se basant sur ce paramètre. La rotation Hourly effectue une rotation par heure, daily - un par jour, weekly - une par semaine et monthly - une par mois. Le moment précis de rotation est défini par une combinaison des variables `RotateDay` et `RotateTime`. Pendant

la rotation, un fichier existant est renommé en CURRENT_FILENAME.YYYYMMDD-HHMMSS, où YYYYMMDD-HHMMSS est remplacé par l'horodate courant, et les nouvelles lignes sont écrites dans un fichier vide. Pour désactiver cette rotation, ne précisez pas ce paramètre ou fixez le à 0.

Exemple 1 - rotation chaque heure (00 :45, 01 :45, ..., 23 :45) :

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

Exemple 2 - rotation chaque jour à 23 :00 (11PM) :

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

Exemple 3 - rotation chaque Dimanche à 00 :59 :

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

Exemple 4 - rotation le dernier jour de chaque mois :

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

4.2.4 Section [RoutedMode]

Call signalling messages may be passwd in two ways. The first method is Direct Endpoint Call Signalling, in which case the call signalling messages are passed directly between the endpoints. The second method is Gatekeeper Routed Call Signalling. In this method, the call signalling 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 signalling 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 4.2.5 (next section). All settings in this section are affected by reloading.

- `GKRouted=1`
  Default : `0` Whether to enable the gatekeeper routed mode.
- `H245Routed=1`
  Default : `0` Whether to route the H.245 control channel. Only takes effect if `GKRouted=1`.
- `CallSignalPort=0`
  Default : `1721` The port of call signalling 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 call signalling handler. You may increase this number in a heavy loaded gatekeeper. The number can only be increased at runtime. If you have a SMP machine, you can set this number to your number of CPUs.
- `RtpHandlerNumber=2`
  Default : `1` The number of RTP proxy handling threads.
- `AcceptNeighborsCalls=1`
  Default : `1` With this feature enabled, the call signalling 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 signalling address in LCF in responding to an LRQ. That means, the call signalling 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 tunnelling 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.
- `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 signalling 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 signalling 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 signalling channels. Note the range size may limit the number of concurrent calls.
- `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.
- `ConnectTimeout=60000`
  Default : `180000` Timeout value in milliseconds to wait before removing unconnected calls from the call table. This is a guard timer that does not allow unconnected calls to hang forever in the call table. Note that making this value too short may result in calls dropped before answered.
- `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.

4.2.5 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 tunnelling 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 4.2.4 (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.
- `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.

5 Configuration du Routage

Les sections suivantes du fichier de configuration peuvent être utilisées pour configurer comment les appels sont routés.

5.1 Section [RoutingPolicy]

Cette section explique comment les diverses politiques de routage possibles du gatekeeper fonctionnenet.

Les demandes d'appel peuvent être routées en utilisant un certain nombre de fournisseurs de routage :

- `explicit`
  La destination est explicitement indiquée dans la demande de routage.
- `internal`
  La règle classique ; cherche la destination dans la RegistrationTable
- `parent`
  Route l'appel en utilisant des informations envoyées par le GK parent en réponse à un ARQ que le gatekeeper enverra.
- `neighbor`
  Route l'appel en utilisant les voisins en échangeant des messages LRQ
- `dns`
  La destination est résolue par DNS, à condition qu'elle soit résolvable
- `vqueue`
  Utilise le mécanisme de queue virtuelle et génère un événement RouteRequest pour laisser une application externe faire le routage (ne peut être utilisé qu'avec OnARQ)
- `numberanalysis`
  Fournit un support pour l'envoi des chiffres recouverts(?) pour les messages ARQ.

Si une politique ne correspond pas, la politique suivante est essayée.

Ces politiques peuvent être appliquées à un certain nombre de types de requêtes de routage et de données d'entrée de routage. Les différents types sont : ARQ, LRQ, Setup et Facility (avec la raison callForwarded) Il y a aussi la politique de routage générale, qui est une sorte de valeur par défaut pour les autres types.

Exemple :

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

Quand un des messages est reçu qui demande une décision de routage, tous les appels à un alias du type h323_ID seront résolus en utilisant le DNS. Si le DNS échoue à déterminer l'alias, il est comparé à la table interne d'enregistrement. Si un appel est demandé pour un alias commençant par 002, les voisins sont d'abord vérifiés puis la table interne d'enregistrement. Si l'alias demandé n'est par un h323_ID ou un alias commençant par 002, la politique par défaut est utilisée en recherchant dans la table interne d'enregistrement, puis les voisins, et si çà échoue le parent.

Pour les messages ARQ, LRQ, Setup et Facility on peut utiliser les sections [RoutingPolicy : :OnARQ], [RoutingPolicy : :OnLRQ], [RoutingPolicy : :OnSetup] et [RoutingPolicy : :OnFacility] en utilisant la syntaxe expliquée ci-dessus.

Exemple :

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

5.2 Section [RasSrv : :RewriteE164]

Cette section définit les règles de réécriture pour dialedDigits (numéro E.164).

Format :

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

Si le numéro commence avec `original-prefix`, il est réécrit en `target-prefix`. Si le drapeau '`!`' précède le `original-prefix`, le sens est inversé et le target-prfix est ajouté au début du numéro composé. Les caractères jokers spéciaux (`'.'` et `'%'`) sont disponibles.

Exemple :

`08=18888`

Si vous composez `08345718`, il est réécrit en `18888345718`.

Exemple :

`!08=18888`

Si vous composez `09345718`, il est réécrit en `1888809345718`.

Option :

- `Fastmatch=08`
  Default : `N/A` Réécrit uniquement les dialDigits commençant par le préfixe indiqué.

5.3 Section [RasSrv : :GWRewriteE164]

Cette section décrit la réécriture des dialedDigits des numéros E.164 en fonction de la passerelle d'où provient l'appel ou vers où est envoyé l'appel. Ceci permet une manipulation beaucoup plus flexible des dialedDigits pour le routage, etc. En combinaison avec le 5.2 (RasSrv : :RewriteE164) vous pouvez avoir une réécriture en trois étapes :

```
Appel de "gw1", dialedDigits 0867822
                |
                |
                V
Règles d'entrée pour "gw1", dialedDigits now 550867822
                |
                |
                V
Règles générales, dialedDigits maintenant 440867822
                |
                |
                V
Sélection de passerelle, dialedDigits maintenant 440867822, passerelle de sortie "gw2"
                |
                |
                V
Règles de sortie pour "gw2", dialedDigits maintenant 0867822
                |
                |
                V
Appel de "gw2", dialedDigits 0867822
```

Format :

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

Si l'appel correspond à la passerelle, la direction et commence par `original-prefix` il est réécrit en `target-prefix`. Si le drapeau '`!`' précède le `original-prefix`, le sens est inversé. Les caractères jokers spéciaux ('`.`' and '`%`') sont disponibles. Des règles multiple pour une même passerelle doivent être séparées par ';'.

Exemple :

`gw1=in=123=321`

Si un appel est reçu de "gw1" à `12377897`, il est réécrit en `32177897` avant qu'une action supplémentaire ne soit prise.

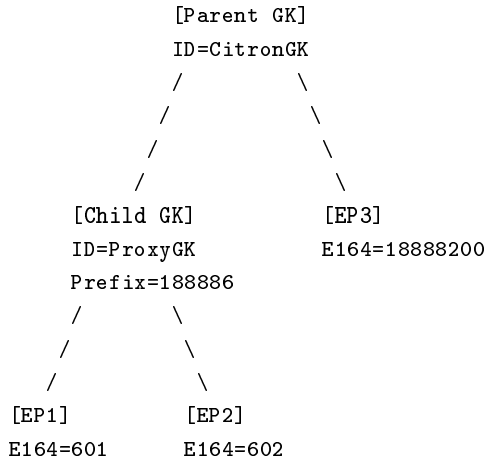
5.4 Section [Endpoint : :RewriteE164]

Une fois que vous avez défini le(s) préfixe(s) pour votre terminal gatekeeper, le gatekeeper parent routera les appels avec **dialedDigits** commençant par ces préfixes. Le gatekeeper fils peut réécrire la destination en accord avec les règles définies dans cette section. Par contraste, quand un terminal interne appelle un terminal enregistré dans le gatekeeper parent, la source sera réécrite à l'envers.

Format :

`external prefix=internal prefix`

Par exemple, si vous avez la configuration suivante,



Avec cette règle :

```
188886=6
```

Quand EP1 appelle EP3 avec `18888200`, le CallingPartyNumber dans le Q.931 Setup sera réécrit en `18888601`. Inversement, EP3 peut atteindre EP1 et EP2 en appelant `18888601` et `18888602`, respectivement. En conséquence, un terminal enregistré auprès du GK fils avec le préfixe '`6`' apparaîtra comme un terminal avec le préfixe '`188886`', pour les terminaux enregistrés auprès du gatekeeper parent.

Cette section ne se rapporte pas à la section 5.2 (RasSrv : :RewriteE164), bien que le plus récent prendra effet en premier.

5.5 Section [Routing : :NumberAnalysis]

Cette section définit les règles pour la politique de routage `numberanalysis`. La politique contrôle un numéro composé pour un nombre minimum et/ou maximum de chiffres et envoie ARJ, si nécessaire (le nombre de chiffres est en dehors de la plage), pour supporter l'envoi de chiffres qui se recouvrent.

Format :

`prefix=MIN_DIGITS[ :MAX_DIGITS]`

Si le numéro correspond au `prefix`, il est vérifié qu'il est composé d'au moins `MIN_DIGITS` chiffres et (si MAX_DIGITS est présent) d'au plus `MAX_DIGITS` chiffres. Les caractères joker spéciaux (`!`, '`.`' et '`%`') sont disponibles. Si le numéro est trop court, un ARJ est envoyé avec `rejectReason` fixé à `incompleteAddress`. Si le numéro est trop long, un ARJ est envoyé avec `rejectReason` fixé à `undefinedReason`. La liste de préfixes est parcourue du préfixe le plus long au plus court pour une correspondance.

Exemple :

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

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

Les appels aux destinations commençant par 0048 ont besoin d'au moins 12 chiffres, par 48 - 10 chiffres et pour tous les autres au moins 6 et au plus 20 chiffres.

6 Configuration RAS

6.1 Section [RasSrv : :GWPrefixes]

Cette section liste quels numéros E.164 sont routés vers une passerelle spécifique.

Format :

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

Il faut noter que vous devez indiquer l'alias de la passerelle. Si une passerelle est enregistrée pour cet alias, tous les numéros commençant par les préfixes sont routés vers cette passerelle. Les caractères spéciaux . et ! peuvent être utilisés ici pour correspondre à n'importe quel chiffre et désactiver le préfixe.

Exemple :

`test-gw=02,03`

6.2 Section [RasSrv : :PermanentEndpoints]

Dans cette section vous pouvez mettre des terminaux qui n'ont pas le support RAS ou que vous ne souhaitez pas voir expirer. Les enregistrements seront toujours conservés dans le table d'enregistrements du gatekeeper. Cependant, vous pouvez toujours le désenregistrer par le port d'état. Les caractères spéciaux . et ! peuvent être utilisés avec les préfixes ici pour correspondre à n'importe quel chiffre et désactiver le préfixe.

Format :

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

Exemple :

Pour une passerelle,

`10.0.1.5=Citron ;009,008`

Pour un terminal,

`10.0.1.10 :1720=700`

6.3 Section [RasSrv : :RRQFeatures]

- `AcceptEndpointIdentifier=1`
  Défault : `1` Indique si on accepte **endpointIdentifier** indiqué dans un RRQ complet.
- `AcceptGatewayPrefixes=1`
  Défault : `1` Une passerelle peut enregistrer ses préfixes avec le gatekeeper en contenant **supportedPrefixes** dans le champ **terminalType** du RRQ. Cette option indique si on accepte les préfixes de cette passerelle.
- `OverwriteEPOnSameAddress=1`
  Défault : `0` Dans certains réseaux l'adresse IP d'un terminal peut changer de façon inattendue. Ceci peut se produire quand un terminal utilise une connexion PPP (e.g. modem ou ADSL). Cette option indique comment gérer un demande d'enregistrement (RRQ) d'une adresse IP qui ne correspond pas à ce que nous avons stocké. L'action par défaut est de rejeter la requête. Avec cette option activée la requête contradictoire entraînera une requête non-enregistré (URQ) à être envoyée pour l'adresse IP existante et l'entrée est supprimée pour permettre le terminal de s'enregistrer avec la nouvelle adresse.
- `IRQPollCount=0`
  Défault : `1` Quand le gatekeeper ne reçoit pas un RRQ garder-en-vie d'un terminal pendant une période de temps TimeToLive, il envoie un message IRQ pour "sonder" le terminal et vérifier si il est en vie. Après que IRQPollCount messages aient été envoyés sans qu'aucune réponse n'ait été reçue, le terminal est désenregistré. Pour désactiver cette fonction (et désenregistrer les terminaux juste aprés TimeToLive), régler cette variable à 0. L'intervalle de sondage IRQ est de 60 secondes.

6.4 Section [RasSrv : :ARQFeatures]

- `ArjReasonRouteCallToSCN=0`
  Défault : `1` Si oui, le gatekeeper rejette un appel d'une passerelle à elle-même avec la raison **routeCallToSCN**.
- `ArjReasonRouteCallToGatekeeper=1`
  Défault : `1` Si oui, le gatekeeper rejette un ARQ répondu sans un CallRec préexistant dans la CallTable avec la raison **routeCallToGatekeeper** en mode routé. Le terminal doit libérer l'appel immédiatement et renvoyer l'appel Setup au gatekeeper.
- `CallUnregisteredEndpoints=0`
  Défault : `1` Avec cette option activée, le gatekeeper acceptera un ARQ d'un terminal enregistré avec **destCallSignalAddress**, que l'adresse appartienne à un terminal enregistré ou non. Ceci signifie que vous pouvez explicitement indiquer l'IP du terminal (enregistré ou non) que vous voulez appeler.
- `RemoveTrailingChar=#`
  Défault : `N/A` Indique le caractère de fin à supprimer de **destinationInfo**. Par exemple, si votre terminal contient de manière erronée le caractère de fin tel que '#' dans **destinationInfo**, vous pouvez le supprimer avec cette option.
- `RoundRobinGateways=0`
  Défault : `1` Active/désactive la sélection round-robin(?) de passerelle, si plus d'une passerelle correspond à un numéro composé. Si désactivé, la première passerelle disponible sera sélectionnée. Autrement, les appels suivants seront envoyés à chaque passerelle à son tour.

6.5 Section [NATedEndpoints]

Le gatekeeper peut détecter automatiquement si un terminal est derrière un routeur. Cependant, si la détection échoue, vous pouvez l'indiquer manuellement dans cette section.

Format :

```
alias=true,yes,1,...
```

Exemple :

Indique qu'un terminal avec l'alias 601 est derrière un routeur.

```
601=true
```

7 Configuration de l'Authentication

La section suivante du fichier de configuration permet de configurer l'authentification.

7.1 Section [Gatekeeper : :Auth]

La section définie le méchanisme d'authentification du gatekeeper.

Syntax :

```
authrule=actions

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

Une règle peut retourner un de ces trois codes : ok, fail, pass.
- `ok` - La requête est authentifié par le module.
- `fail` - L'authentification a echoué et doit être rejeté.
- `next` - La règle ne permet pas d'authentifié la requête.

Il existe aussi trois façons de contrôler une règle.
- `optional` - Si la règle ne peut authentifié la requête, on passe à la règle suivante.
- `required` - Les requêtes doivent être authentifiées par ce module, ou elles seront rejetées. Une fois authentifiée par ce module, la requête passera à la règle suivante.
- `sufficient` - Si la règle est authentifié, elle est acceptée, sinon elle est rejetée. Aucune règle ne devrait suivre une règle 'sufficient' car elle ne servira jamais.

Modules actuellement supportés :
- `SimplePasswordAuth/SQLPasswordAuth/LDAPPasswordAuth` Ces modules vérifient les champs **tokens** or **cryptoTokens** du message RAS. Les champs doivent au moins contenir generalID and password. Le hashage simple MD5 des champs **cryptoTokens**, **cryptoEPPwdHash** et le hashage HMAC-SHA1-96 (libssl doit être installé) du champ **nestedcryptoToken** sont maintenant supportés. les hashages des champs par CAT (Cisco Access Token) et en texte clair (username/password) sont aussi supportés. Le ID et le password sont lus depuis la section 7.2 ([SimplePasswordAuth]), une base de données SQL ou LDAP respectivement par les modules `SimplePasswordAuth`, `SQLPasswordAuth` et `LDAPPasswordAuth`. Le module `MySQLPasswordAuth` afin de conserver la compatibilité avec les versions précédentes.
- `AliasAuth/SQLAliasAuth/LDAPAliasAuth` Ce module ne peut être utilisé que pour l'authentification des RegistrationRequest (RRQ). L'adresse IP d'un terminal et un alias donné doit vérifier une pattern donnée. Pour `AliasAuth`, la pattern est définie dans la section 7.4 ([RasSrv : :RRQAuth]). Pour `SQLAliasAuth`, la pattern est rappatrier à partir d'une base de données SQL, définie dans la section 7.5 ([SQLAliasAuth]). Pour `LDAPAliasAuth`, l'alias (par défaut : mail attribute) et l'adresse IP (par défaut : voIPIpAddress attribute) doivent être trouvé dans une entrée LDAP.
- `PrefixAuth` A l'origine ce module se nommait `GkAuthorize`. L'adresse IP et l'alias de la requête et un préfix donné doivent vérifier une pattern donnée. Pour plus de détails voir la section 7.6 ([PrefixAuth]). Actuellement le module ne sait vérifier que les AdmissionRequest (ARQ) et les LocationRequest (LRQ).
- `RadAuth` Ce module permet l'authentification suivant le schéma de sécurité username/password d'H.235. Il authentifie RRQ, ARQ, Q931 Setup grâce à des serveurs RADIUS distants. Il envoie aux serveurs RADIUS les usernames/passwords extrait du **tokens** CAT (Cisco Access Tokens) se trouvant dans les paquets RRQ, ARQ et Setup. Donc si votre terminal ne supporte pas les CATs ou bien you n'avez pas besoin d'un schéma d'authentification basé des usernames/password individuels - ce n'est pas le module qu'il vous faut (vous pouvez regarder du côté du `RadAliasAuth`). Pour plus de détails se référer à la section 7.7 ([RadAuth]).
- `RadAliasAuth` Ce module permet d'authentifier à partir des alias de terminals et/ou des adresses IP "call signalling" avec l'aide de serveurs RADIUS. Il ne nécessite aucun **tokens** H.235 dans les messages RAS, il peut donc être uilisé avec une gamme plus large de systemes contrairement au module `RadAuth`. Les messages RRQ, ARQ, et Q.931 Setup peuvent être authentifiés avec ce module. Pour plus de détails se référer à la section 7.8 ([RadAliasAuth]).

Vous pouvez aussi configurer une règle afin de vérifier seulement une partie du message RAS. L'exemple suivant montre comment vérifier uniquement les RRQ et ARQ à partir du module `SimplePasswordAuth` avec une règle optionnelle. Si un RRQ n'est pas vérifié (ne contient pas de champs **tokens** ou **cryptoTokens**), il est passé à la règle suivante `AliasAuth`. La règle par défaut est d'acepter toutes les requêtes.

Exemple 1 :

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

L'exemple ci-dessous authentifie tous les appels, vérifie le détils des messages de signalisation Setup avec le module RadAliasAuth.

Exemple 2 :

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

L'exemple suivant vérifie l'enregistrement des terminaux (RRQ) et l'admission d'appel (ARQ) soit par username/password (RadAuth) soit par alias/IP (RadAuthAlias). De plus, si l'ppel provient d'un terminal non enregistré (ET qu'aucune authentification par RRQ ou ARQ n'est été faite), le module RadAliasAuth vérifie alors le message Setup (SetupUnreg).

Exemple 3 :

```
RadAuth=optional ;RRQ,ARQ
RadAliasAuth=required ;RRQ,ARQ,SetupUnreg
default=allow
```

7.2 Section [SimplePasswordAuth]

Cette section définie la paire userid/password utilisé par le module `SimplePasswordAuth`. Tous les mots de passe sont cryptés avec le programme `addpasswd`.

Usage :

```
addpasswd config section userid password
```

Options :

- `KeyFilled=123`
  Défaut : `0` Valeur par défault à utiliser comme octets de remplissage pendant l'encryption/décryption du mot de passe.
- `CheckID=1`
  Défautt : `0` Vérifie si les alias correspondent aux ID dans les marques.
- `PasswordTimeout=120`
  Défaut : `-1` Le module `SimplePasswordAuth` et tous ses descandants mettent en cache un mot de passe authentifié. Ce champ définie le durée de vie du cache en seconde. `0` signifie qu'aucun cache ne sera utilisé, alors qu'une valeur négative indique que le cache n'expirera jamais.

7.3 Section [SQLPasswordAuth]

Authentifie les terminaux compatibles H.235 à partir de mots de passe enregistrés dans une base de données SQL. Cette section définie les gestionnaires SQL à utiliser, les paramètres de connexion à la base de données SQL et la requête pour récupérer les mots de passe.

- `Driver=MySQL | PostgreSQL`
  Défaut : N/A Gestionnaire de base de données SQL à utiliser. Actuellement seul `MySQL` et `PostgreSQL` sont implémentés.
- `Host=DNS[ :PORT] | IP[ :PORT]`
  Défaut : `localhost` Adresse de l'hôte hébergeant la base de données SQL. Peut-être de la forme de `DNS[ :PORT]` ou `IP[ :PORT]`. Par exemple `sql.mycompany.com` ou `sql.mycompany.com :3306` ou `192.168.3.100`.
- `Database=billing`
  Défault : `billing` Nom de la base de données à laquelle se connecter.
- `Username=gnugk`
  Compte utilisé pour se connecter à la base de données.
- `Password=secret`
  Mot de passe utilisé pour se connecter à la base de données. Si le mot de passe n'est pas indiqué, une

connexion sans mot de passe sera tenter à la base de données. Si `EncryptAllPasswords` est actif, ou si la variable `KeyFilled` est définie dans cette section, le mot de passe sera sous une forme encrypté et doit être créé avec l'utilitaire `addpasswd`.

- `CacheTimeout=120`
  Défaut : `0` Ce champ définie combien de temps la paire (alias :mot de passe) récupérée de la base de données sera conservée en cache en local. L'unité de base est la seconde. `0` signifie que le cache n'est pas actif, une valeur négative désactive l'expiration du cache (seule la commande `reload` forcera le rafraîchissement du cache).
- `MinPoolSize=5`
  Défaut : `1` Définie le nombre de connexions SQL actives simultannées. Cela permet des performances accrues lors de charges élevées, car plusieurs requêtes concurrentes peuvent alors être exécutées en même temps. Si `MinPoolSize=1` le précédent comportement est recréé, les requêtes à la base de données sont sérialisées.
- `Query=SELECT ...`
  Défaut : `N/A` Définie la requête SQL utilisée pour récupérer le mot de passe H.235 de la base de données. La requête est personnalisable - avant chaque requête les nouvelles valeurs sont substituées aux paramètres de remplacement. 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` - l'alias pour lequel on demande le mot de passe
  - `%2` - l'identification de la gatekeeper

  Exemples :

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

7.4 Section [RasSrv : :RRQAuth]

Spécifie l'action lors de la réception d'un message RRQ (confirmation ou rejet) pour le module `AliasAuth`. Le premier alias (le plus souvent un H323ID) du terminal à enregistrer est recherché dans la section. Si un paramètre est trouvé la valeur sera appliquée comme une règle. Une règle est constituée de conditions séparées par des '&'. Un enregistrement est accepté si toutes les conditions sont vérifiées.

Syntax :

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

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

La notation et la signification des <`authparams`> dépendent de <`authtype`> :

- `sigaddr` - expresson régulière étendue qui s'applique à "PrintOn(ostream)", la représentation de l'adresse du signal de la requête. Exemple :

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

- `sigip` - forme spécialisée de '`sigaddr`'. Ecrire l'adresse IP de la signalisation en notation décimal : "`byteA.byteB.byteC.byteD :port`". Exemple :

```
sigip:192.168.242.165:1720
```

- `allow` - toujours accepter l'alias.
- `deny` - toujours rejeter l'alias.

7.5 Section [SQLAliasAuth]

Authentifie les terminaux par des règles stockées dans base de données SQL (les règles doivent être conformes au format de la section 7.4 ([RasSrv : :RRQAuth])). Cette section définie le gestionnaire SQL à utiliser, les paramètres de la connexion à la base de données and la requête à utiliser.

- `Driver=MySQL | PostgreSQL`
  Défaut : N/A Gestionnaire de base de données à utiliser, les versions pour `MySQL` et `PostgreSQL` sont implémentées.
- `Host=DNS[ :PORT] | IP[ :PORT]`
  Défaut : `localhost` Adresse de l'hôte hébergeant le serveur SQL. Peut être de la forme `DNS[ :PORT]` ou `IP[ :PORT]`. Par exemple `sql.mycompany.com` ou `sql.mycompany.com :3306` ou `192.168.3.100`.
- `Database=billing`
  Défaut : `billing` Nom de la base de données à laquelle se connecter.
- `Username=gnugk`
  Compte utilisé pour se connecter à la base de données.
- `Password=secret`
  Mot de passe utilisé pour se connecter à la base de données. Si aucun mot de passe n'est spécifié, une connexion sans mot de passe sera effectuée. Si `EncryptAllPasswords` est selectionné, ou la variable `KeyFilled` est définie dans cette section, le mot de passe sera sous une forme encryptée et peut être généré par l'utilitaire `addpasswd`.
- `CacheTimeout=120`
  Défaut : `0` Ce champ définie combien de temps le couple (alias ;authrule) récupéré depuis la base de données sera conservé dans le cache local. La valeur à pour unité la seconde. `0` signifie qu'aucune règle ne sera mise en cache, et une valeur négative indique que le cache n'expire jamais (seule la commande `reload` forcera le rafraîchissement du cache).
- `MinPoolSize=5`
  Défaut : `1` Définit le nombre de connexions simultanéés à la base de données SQL. Cela permet une performance accrue sous charge élevée (pusieurs requêtes sont alors exécutées en parallèle). Si `MinPoolSize=1` simule le conportement des versions précédentes (les requêtes sont lancées en série, une requête à la fois).
- `Query=SELECT ...`
  Défaut : `N/A` Définie la requête SQL utilisées pour récupérer la règle d'alias de la base de données. La requête est personnalisable - ce qui signifie que le remplacement des paramètres est fait à chaque nouvelle requête. 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` - l'alias pour lequel on recherche la règle.
  - `%2` - l'identification de la gatekeeper.

  Exemple :

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

7.6 Section [PrefixAuth]

Cette section définie les règles d'authentification pour le module `PrefixAuth`. Actuellement, seuls les messages ARQs et LRQs peuvent être authentifiés par ce module.

On débute par le préfix le plus spécifique du champ **destinationInfo** de la reqête reçue. Ensuite la requête est eccepté ou non suivant les règles dans l'ordre spécifique décroissant. Si aucune correspondance de préfix n'est trouvée, et que l'option par `default` est spécifié, cette dernière accepte ou non la reqête. Autrement la requête continue sont parcourt dans les modules d'authification suivant les conditions du module.

Format :

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

Syntax :

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

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

Où <iprule> peut être noté en notation décimal ou CIDR, <aliasrule> est une expression régulière. Si '!' précède la règle, le sens est inversé.

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
```

Dans cette configuration, tous les terminaux sauf ceux du réseau 10.0.0.0/27 sont autorisés à appeler le préfix 555 (execpté le 5555). Les terminaux du réseau 192.168.1.0/24 sont autorisés à appeler le préfix 5555, sauf 192.168.1.1. Les terminaux ne provenant pas de 172.16.0.0/24 ne sont pas autorisés à appeler le préfix 86. Les terminaux ayant un alias commençant par 188884 ne sont pas autorisés à appeler le préfix 09. Toutes les autres situations sont autorisées.

7.7 Section [RadAuth]

Cette section définie les options qui permettent d'activer l'authentification RADIUS basé sur le H.235 CATs (Cisco Access Tokens) présent dans les requêtes RAS RRQ et ARQ et les messages Setup Q931.

- Servers=SERVER1[:AUTH_PORT[:ACCT_PORT[:SECRET]]] ;SERVER2[:AUTH_PORT[:ACCT_PORT[:SECRET]]] ;..
  Default : N/A Définie les serveurs RADIUS utilisés pour l'authentification. Cette liste peut contenir un nombre arbitraire de serveurs. L'ordre des serveurs sera respecté par le module RADIUS lors du processus d'interrogation. Si aucun port UDP n'est indiqué, DefaultAuthPort sera utilisé. De même avec le secret et SharedSecret. Adresses IP ou nom DNS peuvent être utilisés comme nom pour les serveurs RADIUS.

  Exemple Servers **:**

  ```
  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
  Défaut : N/A Interface réseau que le client RADIUS doit utiliser pour communiquer avec les serveurs RADIUS. Ce paramètre peut être utile sur les machines "NATées". By défaut cette valeur est vide et c'est la table de routage de l'OS qui choisit l'adresse IP source. Si vous n'êtes pas certain de ce que vous faîtes, il est préférable de laisser cette option vide.
- RadiusPortRange=10000-11000
  Défaut : N/A Par défaut (si cette option n'est pas définie), le client RADIUS alloue des ports dynamiquement comme indiqué par l'OS. Si vous désirez que le client RADIUS utilise des ports d'une plage particulière seulement - configurer ce paramètre.

- `DefaultAuthPort=PORT_NO`
  Défaut : `1812` Ports UDP à utiliser pour les requêtes d'authentification RADIUS (paquets Access-Request), l'attribut `Servers` est prioritaire.
- `SharedSecret=SECRET`
  Défaut : `N/A (chaîne vide)` Secret utilisé pour authentifier ce GnuGK (client NAS) auprès d'un serveur RADIUS. Cela peut être un mot de passe à fort encryption. C'est la valeur par défaut, si aucun secret spécifique n'est indiqué dans la partie `Servers`. Si `EncryptAllPasswords` est selectionné, or la variable `KeyFilled` est définie dans cette section, alors le mot de passe est sous une forme cryptée et peut être crée à partir de l'utilitaire `addpasswd`.
- `RequestTimeout=TIMEOUT_MS`
  Défaut : `2000` (milisecondes) Délai (en millisecondes) que la réponse du serveur RADIUS à une requête envoyée par GnuGK ne peut dépasser. Si il n'y a pas de réponse dans cette période, une nouvelle requête est envoyé au serveur RADIUS suivant.
- `IdCacheTimeout=TIMEOUT_MS`
  Défaut : `9000` (milisecondes) Timeout (miliseconds) 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 approximatelly 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 exhauted - 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`
  Défaut : `60000` (milisecondes) - 60 s Délai au bout duquel les sockets RADIUS inutilisées sont fermées. Utilisé en conjonction de `IdCacheTimeout` - les sockets supplémentaires créées pendant les périodes où GnuGK est fortement chargé afin de répondre aux requêtes entrantes, sont fermées pendant les périodes d'inactivités.
- `RequestRetransmissions=NUMBER`
  Défaut : `2` Combien de fois une même requête RADIUS est envoyée à chaque serveur RADIUS configurés (si aucune réponse n'est reçue). 1 signifie aucune retransmission, 2 - une seule, ... la méthode exacte de retransmission est définie par l'attribut `RoundRobinServers`.
- `RoundRobinServers=BOOLEAN`
  Défaut : `1` Méthode de retransmission des requêtes RADIUS.
  Si définie à 1, la requête RADIUS est transmise de la façon suivante (jusqu'à ce qu'une réponse soit reçue) :

  ```
  Serveur #1 Essai #1, Serveur #2 Essai #1, ..., Serveur #N Essai #1
  ...
  Serveur #1 Essai #RequêtesRetransmises, ..., Serveur #1 Essai #RequêtesRetransmises
  ```

  Si définie à 0, la séquence suivante est déroulée :

  ```
  Serveur #1 Essai #1, ..., Serveur #1 Essai #RequêtesRetransmises
  ...
  Serveur #N Essai #1, ..., Serveur #N Essai #RequêtesRetransmises
  ```

- `AppendCiscoAttributes=BOOLEAN`
  Défaut : `0` Si définie, les attributs RADIUS spécifiques Cisco (en anglais : VendorSpecificAttribut) sont inclus dans les requêtes RADIUS(h323-conf-id,h323-call-origin,h323-call-type).
- `IncludeTerminalAliases=BOOLEAN`
  Défaut : `1` Si définie, l'attribut VSA Cisco 'h323-ivr-out' est envoyé avec la liste d'alias avec laquelle le terminal s'est enregistré(RRQ.m_terminalAlias). Cet attribut est fourni afin de permettre un contrôle fin sur la liste d'alias du terminal sous lesquels il a le droit de s'enregistrer. Le formadt e cet attribut est :

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

– `UseDialedNumber=BOOLEAN`
Défaut : `0` `UseDialedNumber=1` : renseigne le nombre Called-Station-Id avec le numéro originellement appelé par l'utilisateur. `UseDialedNumber=0` : le renseigne avec le numéro réécri.

7.8 Section [RadAliasAuth]

Cette section permet de configurer la partie qui s'occupe d'activer les authentification RADIUS des alias de terminal ou des adresses IP présents dans les requêtes RAS RRQ, ARQ ou Setup Q.931. Ce schéma authentification est utile à la fois pour les terminaux enregistrés auprès de la GateKeeper (ARQ,RRQ) et aussi pour les appels provenant de terminaux non-enregistrés (Setup).

– `Servers=SERVER1[ :AUTH_PORT[ :ACCT_PORT[ :SECRET]]] ;SERVER2[ :AUTH_PORT[ :ACCT_PORT[ :SECRET]]] ;.`
Défaut : `N/A` Serveurs RADIUS à utiliser pour les requêtes d'authentifcation RAS. Cette liste peut contenir un nombre arbitraire de serveurs. L'ordre des serveurs sera suivi par celui des requêtes. Si aucun port UDP n'est spécifié, c'est la valeur `DefaultAuthPort` qui est utilisée. Si aucun secret est indiqué, la valeur de `SharedSecret` sera utilisée. Des adresses IP comme des noms DNS peuvent être indiqués pour les serveurs.

Exemple :

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

– `LocalInterface=IP_OR_FQDN`
Défaut : `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 overriden by `Servers` attribute.
– `SharedSecret=SECRET`
Default : `N/A (empty string)` Secret used to authenticate this GNU GK (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` (miliseconds) Timeout (miliseconds) for RADIUS server response to a request sent by GNU GK. If no response is received within this time period, next RADIUS server is queried.
– `IdCacheTimeout=TIMEOUT_MS`
Default : `9000` (miliseconds) Timeout (miliseconds) 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 approximatelly 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 exhauted - 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` (miliseconds) - 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 transmissed 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 attibutes 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 behavious 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 Accounting Configuration

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

8.1 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 7.1 ([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 | ...
<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 11.1 ([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 8.2 ([FileAcct]) section.
– `RadAcct` This module performs RADIUS accounting. It supports all event types (start, stop, update, on, off). See the section 8.3 ([RadAcct]) for configuration details.
– `SQLAcct` This module performs direct SQL accounting. It supports (start, stop, update) event types. See the section 8.4 ([SQLAcct]) for configuration details.
– `default` This is a special pseudo module - it is used to set the final status if preceeding 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.

8.2 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 paramterized 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)
  - `%c` - Q.931 disconnect cause (decimal integer)
  - `%s` - unique (for this gatekeeper) session identifier (Acct-Session-Id)
  - `%u` - H.323 ID of the calling party
  - `%{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
  - `%{connect-time}` - timestamp string for a call connected event
  - `%{disconnect-time}` - timestamp string for a call disconnect event
  - `%{caller-ip}` - signaling IP addres of the caller
  - `%{caller-port}` - signaling port of the caller
  - `%{callee-ip}` - signaling IP addres 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)
- `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 speficy throusands (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
```

8.3 Section [RadAcct]

This accounting module sends accounting data to a RADIUS server. Module configuration is almost the same as for RADIUS authenticators (see 7.7 ([RadAuth]) and 7.8 ([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 overriden 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` (miliseconds) Timeout (miliseconds) 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` (miliseconds) Timeout (miliseconds) for RADIUS request 8-bit identifiers to be unique.
- `SocketDeleteTimeout=TIMEOUT_MS`
  Default : `60000` (miliseconds) - 60 s Timeout for unused RADIUS sockets to be closed.
- `RequestRetransmissions=NUMBER`
  Default : `2` How many times a single RADIUS request is transmissed 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 attibutes 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`.

8.4 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`
  Default : N/A SQL database driver to use. Currently, `MySQL` and `PostgreSQL` 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)
  - %c - Q.931 disconnect cause (**hexadecimal** integer)
  - %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
  - %{connect-time} - timestamp string for a call connected event
  - %{disconnect-time} - timestamp string for a call disconnect event
  - %{caller-ip} - signaling IP addres of the caller
  - %{caller-port} - signaling port of the caller
  - %{callee-ip} - signaling IP addres 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)

  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 applied.

9 Configuration des Voisins

9.1 Section [RasSrv : :Neighbors]

Si la destination d'un ARQ est inconnue, le gatekeeper envoie des LRQs à ses voisins pour demander si ils ont le terminal destination. Un voisin est sélectionné si un de ses préfixes correspond à la destination ou il a le préfixe "*". Plus d'un préfixe peut être indiqué. Vous pouvez utiliser les caractères spéciaux " ." et " !" comme joker ou pour désactiver un préfixe donné.

Inversement, le gatekeeper répondra uniquement aux LRQs envoyés par des voisins définis dans cette section. Si vous indiquer un préfixe vide, aucun LRQ ne sera envoyé à ce voisin, mais le gatekeeper acceptera les LRQs en provenant. Par préfixe vide on entend un point-virgule ajouté à l'entrée du voisin. Exemple :

```
GK1=192.168.0.5 ;
```

Si vous omettez le point-virgule, les LRQs seront toujours envoyés à ce voisin.

Le champ password est utilisé pour authentifier les LRQs de ce voisin. Se reporter à la section 7.1 ([Gatekeeper : :Auth]) pour de plus amples détails.

La gestion des voisins a changé de façon significative entre la version 2.0 et la version 2.2. Les voisins peuvent être indiqués de deux façons - l'ancienne et la nouvelle.

Entrée dans le vieux format :

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

Exemple :

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

Entrée dans le nouveau format :

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

Exemple :

```
[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
```

Dans le nouveau format la section `[RasSrv : :Neighbors]` indique uniquement les types de gatekeeper et la configuration de chaque voisin est placée dans une section séparée.

9.2 Section [RasSrv : :LRQFeatures]

Définit certains caractéristiques de LRQ et LCF.

- `NeighborTimeout=1`
  Default : `2` Pause en secondes pour attendre les réponses des voisins. Si aucune réponse de tous les voisins apr&323 ; cette pause, le gatekeeper enverra un ARJ au terminal envoyant l'ARQ.
- `ForwardHopCount=2`
  Défault : `N/A` Si le gatekeeper reçoit un LRQ dont la destination est inconnue, il peut transmettre ce message à ces voisins. Quand le gatekeeper reçoit un LRQ et décide que le message doit être transmis à un autre gatekeeper, il commence par décrémenter le champ **hopCount** du LRQ. Si **hopCount** atteint 0, le gatekeeper ne transmettra pas le message. Cette option définit le nombre de gatekeeper au travers desquels un LRQ peut se propager. Il faut noter que ceci n'affecte que l'émetteur du LRQ, pas celui qui transmet. Ce paramètre peut êtra annulé par la configuration d'un voisin donné.
- `AlwaysForwardLRQ=1`
  Défault : `0` Force le gatekeeper à transmettre un LRQ même si il n'y a pas de champ **hopCount** dans le LRQ. Pour éviter les boucles de LRQ, vous devriez utiliser cette option avec prudence. Cette option est utilisée uniquement pour l'ancien format (2.0) de configuration de voisins, le nouveau lit le paramètre depuis une section de configuration spécifique au voisin.
- `AcceptForwardedLRQ=1`
  Défault : `1` Indique si on doit accepter un LRQ transmis depuis les voisins. Ce paramètre peut être annulé par la configuration d'un voisin donné.
- `IncludeDestinationInfoInLCF=0`
  Défault : `1` Le gatekeeper répond avec des LCFs contenant les champs **destinationInfo** et **destinationType**, les alias et type de terminal du terminal de destination. Le gatekeeper voisin peut alors sauver cette information pour supprimer les LRQs ultérieurs. Cependant, certains fournisseurs de gatekeeper font mauvais usage de cette information, entraînant des problèmes d'interopérabilité. Ne désactiver cette option que si vous rencontrez des problèmes de communication avec un gatekeeper tiers.

- `ForwardResponse=0`
  Défault : `0` Si le gatekeeper transmet le message LRQ reçu il peut décider soit de recevoir la réponse LCF ou de la laisser voyager jusqu'à l'émetteur du LRQ. Positionner cette option à 1 si le gatekeeper doit recevoir les messages LCF pour les LRQs transférés. Ce paramètre peut être annulé par la configuration d'un voisin donné.
- `ForwardLRQ=always | never | depends`
  Défault : `depends` Ce paramètre décide si le LRQ devrait être transmis ou non. `always` transmet le LRQ à chaque fois, `never` bloque la transmission de LRQ, `depends` indique au gatekeeper de ne transmettre un LRQ seulement si son hop count est supérieur à 1. Ce paramètre peut être annulé par la configuration d'un voisin donné.

9.2.1 Section [Neighbor : :...]

Les sections commençant par `[Neighbor : :` sont pour la configuration d'un voisin donné.

- `GatekeeperIdentifier=GKID`
  Défault : `N/A` Identifiant du gatekeeper pour ce voisin. Si cette option n'est pas indiquée l'identifiant est obtenu avec la deuxième partie du nom de la section `Neighbor : :`.
- `Host=192.168.1.1`
  Défault : `N/A` Une adresse IP pour ce voisin.
- `Password=secret`
  Défault : `N/A` Un mot de passe à utiliser pour valider les jetons de crypto reçus des LRQs entrants. `Ce n'est pas encore implémenté`.
- `Dynamic=0`
  Défault : `0` 1 signifie que l'adresse IP de ce voisin peut changer.
- `SendPrefixes=004,002 :=1,001 :=2`
  Défault : `N/A` Une liste de préfixes pour lesquels le voisin s'attend à recevoir des LRQs. Si '*' est indiquée, les LRQs seront toujours envoyés à ce voisin. Une priorité peut être donnée à chaque préfixe de chaque voisin (en utilisant la syntaxe :=), ainsi dans le cas de plusieurs LCD reçus de plusieurs voisins, celui avec la priorité la plus élevée sera choisi pour router l'appel. On peut aussi indiquer au gatekeeper d'envoyer un LRQ à ce voisin en se basant sur un type d'alias :
  SendPrefixes=h323_ID,dialedDigits,001

- `AcceptPrefixes=*`
  Défault : `*` Une liste de préfixes que le gatekeeper acceptera dans les LRQs reçs de ce voisin. Si '*' est indiqué, tous les LRQs seront acceptés de ce voisin. On peut aussi indiquer au gatekeeper d'accepter un LRQ de ce voisin en se basant sur un type d'alias :
  AcceptPrefixes=dialedDigits

- `ForwardHopCount=2`
  Défault : `N/A` Si le gatekeeper reçoit un LRQ dont la destination est inconnue, il peut transmettre ce message à ces voisins. Quand le gatekeeper reçoit un LRQ et décide que le message doit être transmis à un autre gatekeeper, il commence par décrémenter le champ **hopCount** du LRQ. Si **hopCount** atteint 0, le gatekeeper ne transmettra pas le message. Cette option définit le nombre de gatekeeper au travers desquels un LRQ peut se propager. Il faut noter que ceci n'affecte que l'émetteur du LRQ, pas celui qui transmet.
- `AcceptForwardedLRQ=1`
  Défault : `1` Indique si on accepte un LRQ transmis par ce voisin.
- `ForwardResponse=0`
  Défault : `0` Si le gatekeeper transmet le message LRQ reçu il peut décider soit de recevoir la réponse LCF ou de la laisser voyager jusqu'à l'émetteur du LRQ. Positionner cette option à 1 si le gatekeeper doit

recevoir les messages LCF pour les LRQs transférés.
- `ForwardLRQ=always | never | depends`
  Défault : `depends` Ce paramètre décide si le LRQ devrait être transmis ou non. `always` transmet le LRQ à chaque fois, `never` bloque la transmission de LRQ, `depends` indique au gatekeeper de ne transmettre un LRQ seulement si son hop count est supérieur à 1.

10 Configuration Par Terminal

En plus des options standards du fichier de configuration, des paramètres de configuration par terminal peuvent être indiqués dans le fichier de configuration. La syntaxe est la suivante :

10.1 Section [EP : :...]

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

`ALIAS` est remplacé par l'alias réel du terminal auquel les paramètres doivent s'appliquer. Actuellement, les options suivantes sont reconnues :

- `Capacity=10`
  Défault : `-1` Capacité d'appel pour un terminal. Par plus de `Capacity` appels simultanés seront envoyés à ce terminal. Dans le cas des passerelles, si plus d'une passerelle correspond au numéro composé, un appel sera envoyé à la première passerelle disponible (qui a de la capacité disponible).
- `GatewayPriority=1`
  Défault : `1` S'applique uniquement aux passerelles. Permet un routage basé sur la priorité dans la cas où plus d'une passerelle correspond à un numéro composé. Plus la valeur est petite, plus la priorité affectée à la passerelle est élevée. Un appel est routé à la première passerelle disponible (qui a de la capacité disponible) avec la priorité la plus élevée (la valeur `GatewayPriority` la plus petite).
- `GatewayPrefixes=0048,0049,0044`
  Défault : `N/A` Préfixes supplémentaires pour cette passerelle. S'applique uniquement aux passerelles. Les caractères spéciaux `.` et `!` peuvent être utilisés pour correspondre à n'importe quel chiffre et désactiver le préfixe.

Exemple :

```
[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
```

Dans cet exemple, les appels seront envoyés à la passerelle `gw1` à moins que sa capacité ne soit totalement utilisée (60 appels simultanés) puis à la passerelle `gw2`.

11 Advanced Configuration

11.1 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 8.1 ([Gatekeeper : :Acct])). In general, the accounting update is to provide backend services with incrementing call duration for connected calls. The default value 0 tells the gatkeeper 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.

11.2 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`.
- `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 signalling 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 signalling 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 IncomingCallProceeding 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 indentifiers 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.

11.3 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.

11.4 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`
Default : N/A SQL database driver to use. Currently, `MySQL` and `PostgreSQL` 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 parametrized - 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 parametrized - 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 parametrized - 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 (idenitifier)

– `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 parametrized - 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 parametrized - that means parameter replacement is made before each query is executed. Parameter placeholders are denoted by **%1**, **%2**, ... strings. Specify %% to embed a percent character before a digit into string (like **%%1**), specify **%{1}** to allow expansion inside complex expressions like **%{1}123**. For `GWPrefixesQuery` only one parameter is defined :
– `%1` - the gatekeeper identifier
It is expected that the query returns zero or more rows of data, with each row consisting of two columns :
– `column at index 0` - gateway alias
– `column at index 1` - gateway prefixes (comma separated)
Sample query strings :

```
GWPrefixesQuery=SELECT gwalias, gwpfx FROM gwprefix WHERE gk = '%1'
```

12 Surveillance du Gatekeeper

12.1 Port d'état

Le port d'état est l'interface externe pour surveiller et contrôler le gatekeeper. Via cette interface, le gatekeeper enverra des messages à propos des appels en cours de tous les clients connectés et recevra des commandes.

Les messages envoyés par le gatekeeper au port d'état sont regroupés en trois **niveaux de trace de sortie** :
– Niveau 0

Notifications de rechargement et réponses directes aux commandes saisies.

– Niveau 1

Notifications de rechargement, réponses directes aux commandes saisies, CDRs et Requêtes de Routage.

– Niveau 2

Trace tout (notification des rechargement, réponses directes aux commandes saisies, CDRs, Requêtes de Routage, RAS, ...). Il s'agit du niveau de trace par **défaut**.

Le client connecté au port d'état peut choisir le niveau de trace par lequel il est intéressé.

L'interface est un simple port TCP (par défaut : 7000), vous pouvez vous connecter avec telnet ou un autre client. Une exemple d'autre client est l'IHM Java, connue sous le nom de GnuGk ACD.

12.1.1 Domaines d'Application

Ce que vous faîtes avec les pouvoirs de l'Interface d'Etat dépend de vous, mais voici quelques idées :
– Surveillance d'Appel
– Surveillande des terminaux enregistrés
– Interface Utilisateur Graphique

Voir GkGUI.

– Routage d'Appel

Voir GnuGk ACD.

– Applications de Facturation

  Analyser les messages CDR et les transmettre à l'application de facturation.

– Interfacer des extensions externes

  Si vous ne voulez pas publier le code source de ces fonctions supplémentaires, publiez juste le noyau de la fonction et interfacez vous avec au travers de l'interface d'état et conservez les parties externes privées.

12.1.2 Exemples

Supposons que vous êtes juste intéressés par les CDRs (enregistrement des détails des appels) et vous voulez les traiter en lot à intervalles réguliers.

Voici un script Perl simple (`gnugk_cdr.pl`) qui lance le gatekeeper et un client très simple pour l'Interface d'Etat et écrit juste les CDRs dans un fichier de log. Vous aurez besoin de le modifier un peu pour qu'il corresponde à vos besoins.

```
#!/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");
}
```

Rappelez-vous qu'il ne s'agit que d'un exemple pour montrer l'utilisation du port d'état. Vous pouvez utiliser le module FileAcct pour tracer les CDRs dans un système de production.

12.1.3 IHM pour le Gatekeeper

Il y a plusieurs Interfaces Home Machine (IHM - GUI) pour le gatekeeper.

- Java GUIDéveloppé par Jan Willamowius. Vous pouvez surveiller les enregistrements et les appels qui passent par le gatekeeper. Un clic droit sur un bouton vous donne un menu déroulant pour ce terminal. Cette IHM fonctionne avec Java 1.0 présent dans la plupart des navigateurs web. Pour des raisons de sécurité l'IHM doit fonctionner comme une application autonome ou être mise à disposition par un serveur web sur le même numéro IP que le gatekeeper (vous ne pouvez pas le lancer en tant qu'applet depuis un fichier local).
  Le programme est disponible à *GnuGk Java GUI* <http://www.gnugk.org/h323gui.html>
- GkGUIUn nouveau programme Java autonome développé par *Citron Network Inc.* <http://www.citron.com.tw/> Il nécessite Java 1.4. Les nouvelles fonctions comprennent :
  - Surveillance de plusieurs gatekeeper simultanément.
  - Deux modes d'affichage : Liste de Boutons ou Arbre.
  - Enregistrement des Détails d'Appel (CDR) et statistiques.
  - GK Status Log.
  - Couleurs différentes pour différents types de terminaux.
  - Modification de la configuration du gatekeeper.
  - Désenregistrement forcé de terminaux.
  - Sauvegarde et imprime le log d'état et CDR.

  Le GkGUI est distribué sous GNU General Public License, disponible à *GnuGk Development* <http://www.gnugk.org/h323develop.html#java>

12.2 Commandes (Référence)

Cette section liste toutes les commandes que vous pouvez émettre sur le port d'état (manuellement ou avec une application externe). Les commandes ne tiennent pas compte de la casse (majuscules / minuscules). Mais certains paramètres peuvent en tenir compte.

La commande `help` ou `h` affichera la liste de toutes les commandes disponibles.

- `Reload` Recharge la configuration.
- `Version`, `v` Afficher la version et des informations sur l'OS du gatekeeper.
- `Statistics`, `s` Affiche des informations statistiques du gatekeeper.

  Exemple :

```
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`, `?` Affiche tous les terminaux enregistrés.

Format :

```
AllRegistrations
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
...
Number of Endpoints: n
;
```

Exemple :

```
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`, `? ?` Affiche des détails sur tous les terminaux enregistrés.

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
;
```

Exemple :

```
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`, `!` Affiche tous les appels en cours en utilisant la même syntaxe ACF que lors de l'établissement de l'appel.

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
;
```

Exemple :

```
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`, `! !` Affiche des détails sur tous les appels en cours.

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
;
```

Exemple :

```
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` Trouve un terminal enregistré par un alias ou un préfixe.

Format :

```
Find Alias
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
;
```

Exemple :

```
f 800
RCF|10.1.1.10:1720|800:dialedDigits=Wei:h323_ID|terminal|1289_endp
;
f 801
SoftPBX: alias 801 not found!
```

– `FindVerbose`, `fv` Trouve des détails d'un terminal enregistré par un alias ou un préfixe.

Format :

```
FindVerbose Alias
RCF|IP:Port|Aliases|Terminal_Type|EndpointID
Registration_Time C(Active_Call/Connected_Call/Total_Call) <r>
[Prefixes: ##] (gateway only)
;
```

Exemple :

```
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` Force l'annulation de l'enregistrement d'un terminal en fonction de son IP et appelle le port de signalisation.

Format :

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

Exemple :

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

– `UnregisterAlias` Force l'annulation de l'enregistrement d'un terminal en fonction d'un de ses alias.

Format :

```
UnregisterAlias Alias
```

Exemple :

```
UnregisterAlias 601
URQ|10.0.1.31:1032|1326_endp|maintenance;
SoftPBX: Endpoint 601 unregistered!
```

– `UnregisterAllEndpoints` Force l'annulation de l'enregistrement de tous les terminaux enregistrés.

Format :

Exemple :

```
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` Déconnecte un appel avec le numéro donné.

Format :

```
DisconnectCall Number
```

Exemple :

```
DisconnectCall 1533
```

– `DisconnectIP` Déconnecte tous les appels d'un terminal en fonction de son IP et appelle le port de signalisation.

Format :

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

Exemple :

```
DisconnectIP 10.0.1.31:1720
```

– `DisconnectAlias` Déconnecte tous les appels d'un terminal en fonction d'un de ses alias.

Format :

```
DisconnectAlias Alias
```

Exemple :

```
DisconnectAlias 601
```

– `ClearCalls` Déconnecte tous les appels sur le gatekeeper.
– `GK` Affiche les informations sur le gatekeeper parent.
– `Trace` Règle le niveau de trace de sortie de l'interface d'état. Il contrôle quels messages sont envoyés à ce client :
  – `trace 0` ou `trace min` Seulement les réponses directes aux commandes et les notifications de rechargement
  – `trace 1` CDRs, réponses directes aux commandes et notifications de rechargement.
  – `trace 2` ou `trace max` Affiche tout (RAS, CDRs, réponses directes aux commandes, notifications de rechargement, etc).
– `Debug` Utilisé uniquement à de buts de debug. Options :
  – `trc [+|-|n]` Affiche/modifie le niveau de trace.
  – `cfg SEC PAR` Lit et imprime un paramètre de configuration dans une section.
  – `set SEC PAR VAL` Ecrit une valeur de paramètre de configuration dans une section.
  – `remove SEC PAR` Supprime une valeur de paramètre de configuration dans une section.
  – `remove SEC` Supprime une section.
  – `printrm VERBOSE` Imprime tous les enregistrements de terminaux supprimès.

Exemple :

```
debug trc 3
debug set RoutedMode H245Routed 1
```

– `Who` Affiche toutes les personnes sur le port d'ètat.
– `RouteReject` Termine cette appel sur une file virtuelle. Cette commande est utilisée comme réponse à un événement RouteRequest (voir ci-dessous).

Format :

```
RouteReject CallingEndpointID CallRef
```

Exemple :

```
RouteReject endp_4711 1234
```

– `RouteToAlias`, `rta` Route cet appel sur une file virtuel vers un alias donné. Cette commande est utilisée en réponse à un événement RouteRequest (voir ci-dessous).

Format :

```
RouteToAlias Alias CallingEndpointID CallRef
```

Exemple :

```
RouteToAlias Suzi endp_4711 1234
```

– `RouteToGateway`, `rtg` Route cet appel sur une file virtuelle vers l'alias donné et positionne le destination-SignallAddress. Cette commande est utilisée en réponse à un événement RouteRequest (voir ci-dessous). Vous pouvez utiliser cette commande pour router des appels vers des passerelles hors de la zone ou des MCUs non enregistrés auprès du gatekeeper. Assurez-vous que la politique 'vqueue' et 'explicit' est en cours pour ces appels.

Format :

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

Exemple :

```
RouteToGateway Suzi 192.168.0.50 endp_4711 1234
```

- `Exit`, `q` Quitte le port d'ètat.
- `TransferCall` Transfère un appel étable d'un alias A vers un alias B. Quand l'alias A parlait avec l'alias X, alors l'alias A parle avec l'alias B après le TransferCall.
  Actuellement, ceci ne fonctionne qu'avec les terminaux qui supportent correctement les messages du mécanisme Q.931 (il ne fonctionne donc pas avec Netmeeting).

  Format :

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

  Exemple :

  ```
  TransferCall Frank Peter
  ```

12.3 Messages (Référence)

Cette section décrit la sortie des messages sur l'interface d'état.

- `GCF|IP|Aliases|Endpoint_Type` ; Le gatekeeper reçoit un GatekeeperRequest (GRQ) et répond avec un GatekeeperConfirm (GCF).
- `GRJ|IP|Aliases|Endpoint_Type|RejectReason` ; Le gatekeeper reçoit un GatekeeperRequest (GRQ) et répond avec un GatekeeperReject (GRJ).
- `RCF|IP :Port|Aliases|Endpoint_Type|EndpointID` ; Le gatekeeper reçoit un RegistrationRequest (RRQ) et répond avec un RegistrationConfirm (RCF).
- `RRJ|IP|Aliases|Endpoint_Type|RejectReason` ; Le gatekeeper reçoit un RegistrationRequest (RRQ) et répond avec un RegistrationReject (RRJ).
- `ACF|Caller_IP :Port|Caller_EndpointID|CRV|DestinationInfo|SrcInfo|IsAnswered[|CallID]` ; Le gatekeeper reçoit un AdmissionRequest (ARQ) et répond avec un AdmissionConfirm (ACF). Le CallID est envoyé seulement quand SignalCallId=1.
- `ARJ|Caller_IP :Port|DestinationInfo|SrcInfo|IsAnswered|RejectReason[|CallID]` ; Le gatekeeper reçoit un AdmissionRequest (ARQ) et répond avec un AdmissionReject (ARJ). Le CallID est envoyé seulement quand SignalCallId=1.
- `DCF|IP|EndpointID|CRV|DisengageReason[|CallID]` ; Le gatekeeper reçoit un DisengageRequest (DRQ) et répond avec un DisengageConfirm (DCF). Le CallID est envoyé seulement quand SignalCallId=1.
- `DRJ|IP|EndpointID|CRV|RejectReason[|CallID]` ; Le gatekeeper reçoit un DisengageRequest (DRQ) et répond avec un DisengageReject (DRJ). Le CallID est envoyé seulement quand SignalCallId=1.
- `LCF|IP|EndpointID|DestinationInfo|SrcInfo` ; Le gatekeeper reçoit un LocationRequest (LRQ) et répond avec un LocationConfirm (LCF).
- `LRJ|IP|DestinationInfo|SrcInfo|RejectReason` ; Le gatekeeper reçoit un LocationRequest (LRQ) et répond avec un LocationReject (LRJ).
- `BCF|IP|EndpointID|Bandwidth` ; Le gatekeeper reçoit un BandwidthRequest (BRQ) et répond avec un BandwidthConfirm (BCF).
- `BRJ|IP|EndpointID|Bandwidth|RejectReason` ; Le gatekeeper reçoit un BandwidthRequest (BRQ) et répond avec un BandwidthReject (BRJ).
- `UCF|IP|EndpointID` ; Le gatekeeper reçoit un UnregistrationRequest (URQ) et répond avec un UnregistrationConfirm (UCF).
- `URJ|IP|EndpointID|RejectReason` ; Le gatekeeper reçoit un UnregistrationRequest (URQ) et répond avec un UnregistrationReject (URJ).

- `IRQ|IP :Port|EndpointID ;` Le gatekeeper envoie un InfoRequest (IRQ) à un terminal pour lui demander si il est toujours en vie. Le terminal doit répondre avec un InfoRequestResponse (IRR) immédiatement.
- `URQ|IP :Port|EndpointID|Reason ;` Le gatekeeper envoie un UnregistrationRequest (URQ) à un terminal pour annuler son enregistrement. Le terminal doit répondre avec un UnregistrationConfirm (UCF).
- `CDR|CallNo|CallId|Duration|Starttime|Endtime|CallerIP|CallerEndId| \ CalledIP|CalledEndId|DestinationInfo|SrcInfo|GatekeeperID ;` Après un appel déconnecté, l'enregistrement du détail de l'appel est affiché (sur une ligne).
- `RouteRequest|CallerIP :Port|CallerEndpointId|CallRef|VirtualQueue|CallerAlias[|CallID] ;` Demande à une application externe de router un appel arrivant sur une file virtuelle. Ceci peut être fait avec une commande RouteToAlias ou RouteReject. Le CallID est envoyé seulement quand SignalCallId=1.