InfiNet Session Gateway Controller

System administrator manual

Creation date**: 14 July 2004**

Table of contents

I. Getting started

This document is a description of the InfiNet Wireless Session Gateway Controller system and contains installation guidelines, instructions for the system configuration and maintenance. This manual includes a detailed description of methods and hints which let the user to use the system efficiently.

This manual is aimed for the system administrator who is in charge of SGC setup, configuration and maintenance.

1. Scope of Manual

The document consists of the following chapters:

Getting started

This chapter includes the information about this document purpose and structure.

General description

Contains a description of all basic concepts and ideas that are used in SGC system, gives a product description and its major functional characteristics and usage methods.

Installation

Description of the product installation procedure, hardware/software requirements.

Configuration

This chapter describes the main steps to be taken when configuring the system, subscribers' registration process, termination routes selecting, setting up connections with gatekeepers and other provider's gateways in a very detailed way. A great deal of attention in this chapter will be paid to the peculiarities connected with calls accounting and a billing system interaction.

Configuration file syntax

Contains a detailed description of all commands used for the management and configuration.

Supplemental information

Describes CDR and log files formats, packet formats for the RADIUS-server and other supplemental information.

2. Literature used

1. ITU-T Recommendation H.323, Packet-based multimedia communications systems
2. ITU-T Recommendation H.225.0, Call signalling protocols and media stream packetization for packet-based multimedia communication systems
3. ITU-T Recommendation H.245, Control protocol for multimedia communication
4. ITU-T Recommendation H.235, Security and encryption for H-series (H.323 and other H.245-based) multimedia terminals
5. ITU-T Recommendation H.450.1, Generic functional protocol for the support of supplementary services in H.323

6. ITU-T Recommendation H.450-2, Call transfer supplementary service for H.323
7. ITU-T Recommendation H.450.3, Call diversion supplementary service for H.323
8. ITU-T Recommendation H.450.4, Call hold supplementary service for H.323
9. ITU-T Recommendation H.450.5, Call park and call pickup supplementary services for H.323
10. ITU-T Recommendation H.450.6, Call waiting supplementary service for H.323
11. ITU-T Recommendation H.450.7, Message waiting indication supplementary service for H.323
12. ITU-T Recommendation H.450.8, Name identification supplementary service for H.323
13. RFC 1889 "RTP: A Transport Protocol for Real-Time Applications. Audio-Video"
14. RFC 2865, Remote Authentication Dial In User Service (RADIUS)
15. RFC 2866, RADIUS Accounting

II. General description

Infinet Wireless Session Gateway Controller (SGC) acts as a core element of an IP-telephony provider's system. It implies all major functionality connected with both network's own subscribers (IP-phones and their gateways) and other provider's equipment including authentication, flexible routing and calls accounting. SGC offers a wide range of possibilities including addresses translation and IP-telephony operators numeration plans coordination. SGC supports a contemporary H.323 protocol version 4.

1. Basic concepts

This section describes the terminology and basic vocabulary used in this manual.

Subscriber

A subscriber is an abstract entity which can make and/or receive calls and which can be configured in different ways. Different terminal devices can act as a subscriber (IP-phones, IP-telephony gateways or other gatekeepers). A call is established between two subscribers: '**caller**' and '**callee**'. One subscriber can act in two of these roles simultaneously (for example, when calling from one port of IP-telephony gateway to the other port of the same gateway). For the subscriber description the **user** or the **gatekeeper** directives are used in the SGC configuration (**gatekeeper** directive - for the subscriber with extended functionality). It is not obligatory to associate one subscriber with one physical device. SGC allows using one **user** directive to describe several (even several thousands) of terminals with the help of flexible authentication syntax (including the billing system)

Aliases

Alias is a counterpart of the subscribers' numbers which are used in POTS telehpony. In H.323 this entity is called 'alias' (not 'number') because it has a wider meaning. Thus, in H.323, aliases do not necessarily contain digits but also some other identifiers like strings containing characters can be used (including spaces or other special characters). An E-mail address can be used as an alias. Moreover, the subscriber can be called (or described) with a group of aliases instead of a single one. For example, somebody has an opportunity to make a call to a remote site using any of these aliases: '311212', 'JohnDoe'. In this case, the subscriber which has both of these aliases will be chosen.

Aliases in connection establishment packets

The packets that initiate a connection contain two types of aliases: callee alias and caller alias. In the RAS-packet level - AdmissionRequest - these parameters are placed in DestinationInfo and SrcInfo fields correspondingly. If we are talking about a static subscriber the routing is done using Q.931 fields in the Setup packet. These fields are CalledPartyNumber and DestinationAddress for the callee and CallingPartyNumber and SourceAddress for the caller. All these aliases can be translated by the command **'translate ... alias'**.

2. Functionality description

SGC provides with the following functional features for IP-telephony networks management and monitoring:

Proxy

Proxy possibilities:

- Signaling traffic (H.225, H.245)
- Multimedia traffic (RTP/RTCP)

This feature provides with a possibility to connect inner IP-networks with an outer world totally depersonizing of all crossing traffic and lets to perform precise accounting.

Calls routing

Calls routing is performed with the following parameters to be taken into consideration:

- Aliases (numbers) of a callee and a caller
- Caller membership in some definite group
- Destination gateway attainability
- Destination gateway priority
- Time
- Load balancing within one direction
- Alternate route selecting when priority route faults (call hunting)
- Zones load and their interconnections
- Equipment load

Note *Availability of "Smart" procedure, FastStart or H.245 do not stop the process of alternative subscribers selection. The detailed "Smart" procedure description is available in "**Hunting**" chapter.*

H.323-dialects conversion

SGC is able to establish connections between different manufacturer's devices that are incompatible with each other.

Addresses translation, aliases translation (phone numbers)

Callee and caller aliases translation with **regular expressions** usage lets to:

- coordinate numeration plans of IP-telephony providers with inner numeration plan
- provide the billing system with necessary numbers
- perform flexible routing
- perform subscribers authentication using billing system either

Zone organization and management

SGC allows to make a detailed description of the VoIP-network infrastructure using zones and their interconnections with available bandwidth configuring from zone to zone which will be taken into consideration while authorization and load distribution.

There is a feature available that sets a limit to the number of concurrent calls for the subscriber in the zone or in the link between the zones.

Calls authorization

Authorization can be performed using different methods which include:

- the possibility of a free access (no authorization)
- originator's IP-address
- remote authentication RADIUS service
- call information (e.g. part of the dialed number can be used as a password or subscriber identifier), using the variables of H.235
- H.235 Cisco Access Token (CAT) sending on the RADIUS via Chap-Password

Billing features

Calls billing features:

- RADIUS protocol (including Cisco VSA usage)
- CDR-files
- Calls management features

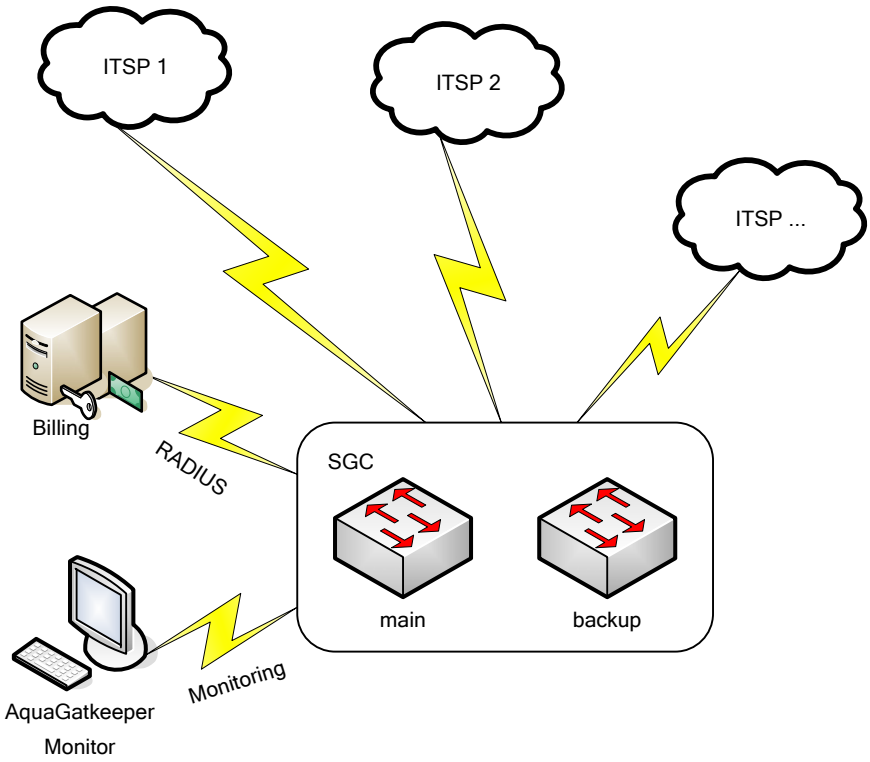
- Current calls list
- Calls detailed information availability
- Forced call drop

3. Usage examples

Listed features provide a wide range of possibilities for SGC usage both for large and small IP-telephony providers. SGC also can be useful for the enterprises that do not deal with IP-telephone services - for their own IP-telephone infrastructure development, remote affiliates connection and for accessing IP-telephone providers (ITSP) services.

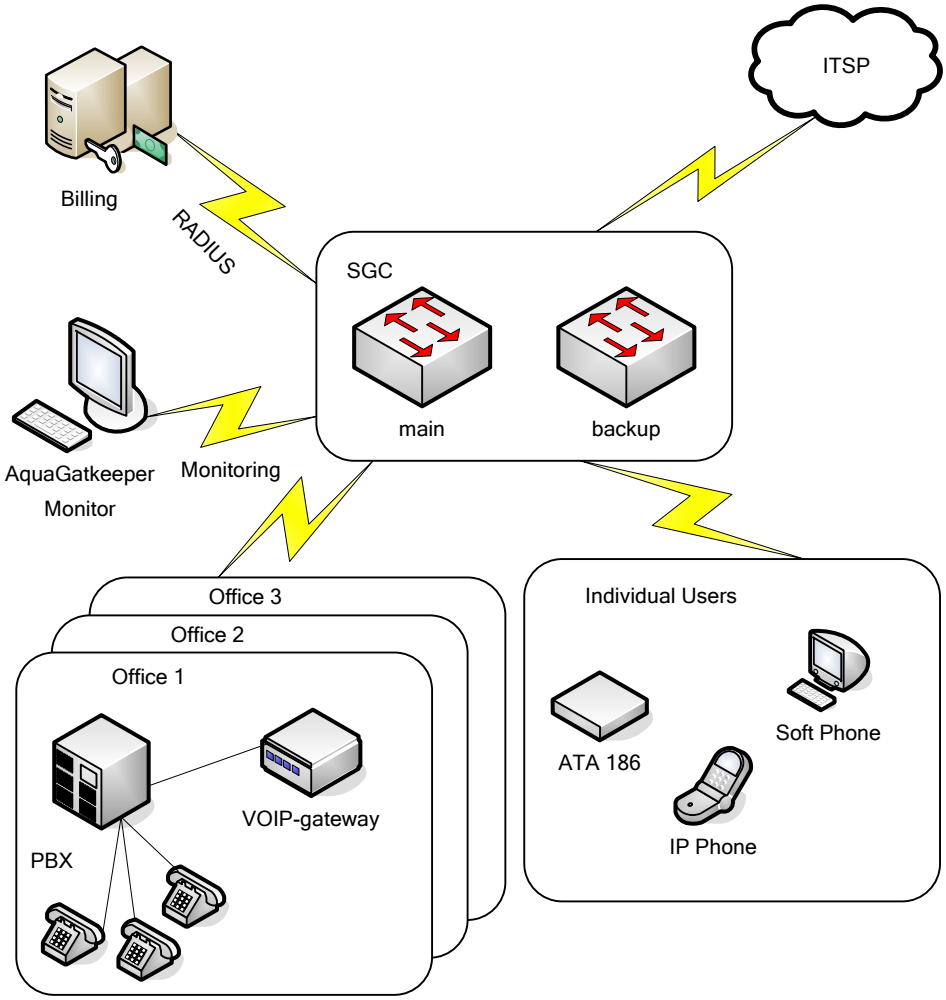
Transit traffic exchange between operators

High overall performance, backup, flexible routing and billing allow organizing VoIP-traffic exchange between IP-telephony operators.



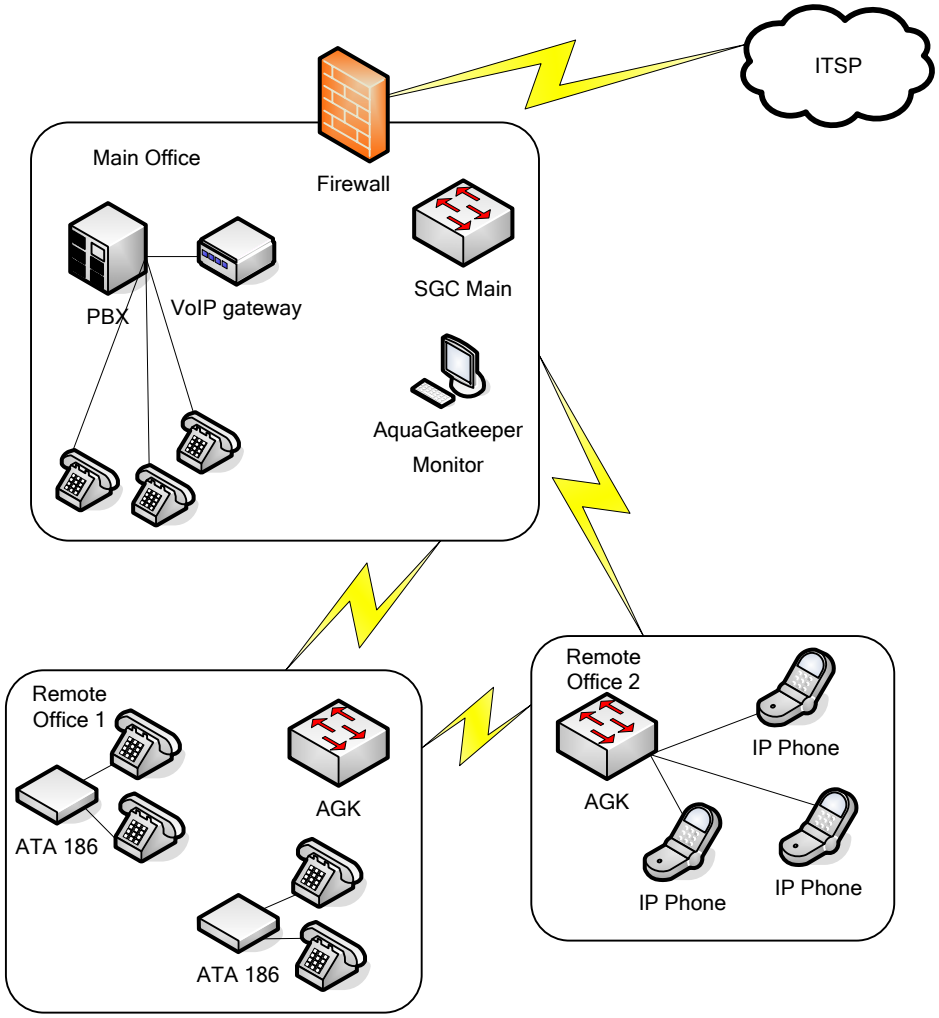
IP-telephony services granting

SGC allows developing the inner numeration plan for each of the corporate clients and at the same time to process individual clients.



Protected private network development

SCG provides with a feature that lets the companies to consolidate its affiliates and to have the only one protected entry point.



III. Installation

SGC has several modifications which work under different operating systems management:

- WANFlex
- FreeBSD 32bit and 64bit
- Linux
- Windows

They do not differ in functional features.

1. WANFlex

For SGC process startup and configuring one should configure WANFlex OS according to the supplied documentation. Further configuring is performed using SGCMonitor.

Configuration stages

- Set up IP-address on eth0 interface
- Create gatekeeper configuration
- Run gatekeeper
- Configure the gatekeeper using SGCMonitor

Example

```
console>ifconfig eth0 192.168.0.1/24 up
console>gatekeeper create
console>gatekeeper start
console>configuration save
```

The device is ready for work and configuring.

2. Unix

SGC system is supplied in an archive. For example, SGC122FreeBSD.tgz file contains SGC version 1.22 for OS FreeBSD.

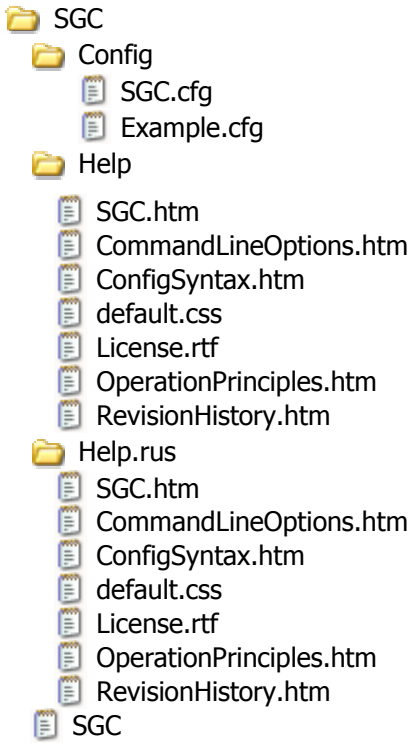
For the software installation it is necessary to extract files from the archive using tar command.

Example

```
tar –xzf SGC122FreeBSD.tgz
```

All files are extracted to the SGC folder which has the following structure.

Structure



File	Description
SGC	Executable image
SGC.cfg	Configuration file

Execution

SGC can work in two modes:

- Console mode
- Unix daemon mode

Console Mode

This mode is not a primary one and serves for the system debugging. In this mode SGC outputs its log directly to the console.

To turn console mode on use the command:

```
./SGC --conf=Config/SGC.cfg
```

Daemon

This is the main mode of SGC (it is also useful when starting up the operating system from **rc scripts**). In this mode, when starting up, SGC process detaches itself from the console and turns into the Unix daemon mode; at that, all log is written to the file specified in the configuration using **log file <filename>** directive; all error-caused events that require special attention are recorded into the system log of the Operating System (syslog).

To run in daemon mode, execute the command:

```
./SGC --daemon --conf=Config/SGC.cfg
```

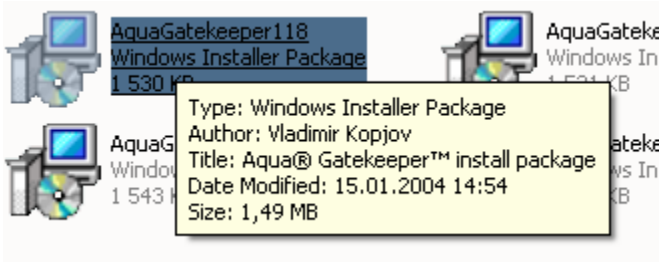
After that the System is ready for work and further configuring.

Note *For the convenient configuration manipulations and SGC monitoring one can install SGC monitor from the installation packet for Windows.*

3. Windows

Installation for OS Windows is performed automatically using build-in component Windows Installer.

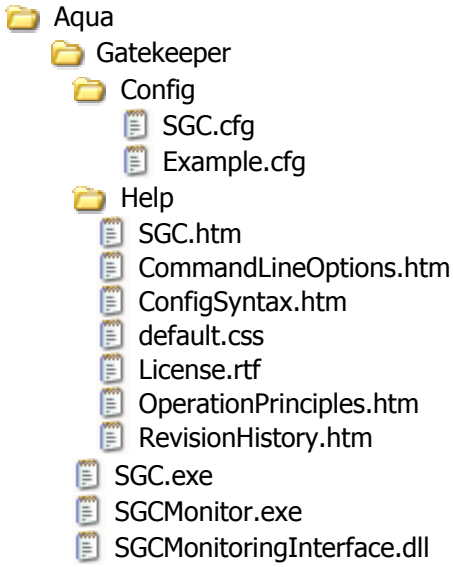
SGC is supplied in "Windows Installer Package" file, for example SGC122.msi and contains SGC and SGCMonitor version 1.22 for OS Windows.



To begin the installation process in the context menu select "Install" item. After that follow the instructions of the "Wizard" to accomplish the installation process.

"Wizard" offers a choice of the destination folder and system components. You can choose either SGC or/and Monitor. After installation is over the system is ready to be run and configured.

Structure



File	Description
SGC.exe	Executable module
SGC.cfg	Configuration file
SGCMonitor.exe	Monitor executable file

Execution

SGC can work in two modes:

- Application (console mode)
- Service (Service)

Application

This mode is not a primary one and serves only for the system debugging. In this mode SGC outputs its log directly to the console window.

To switch to the application mode, use the command:

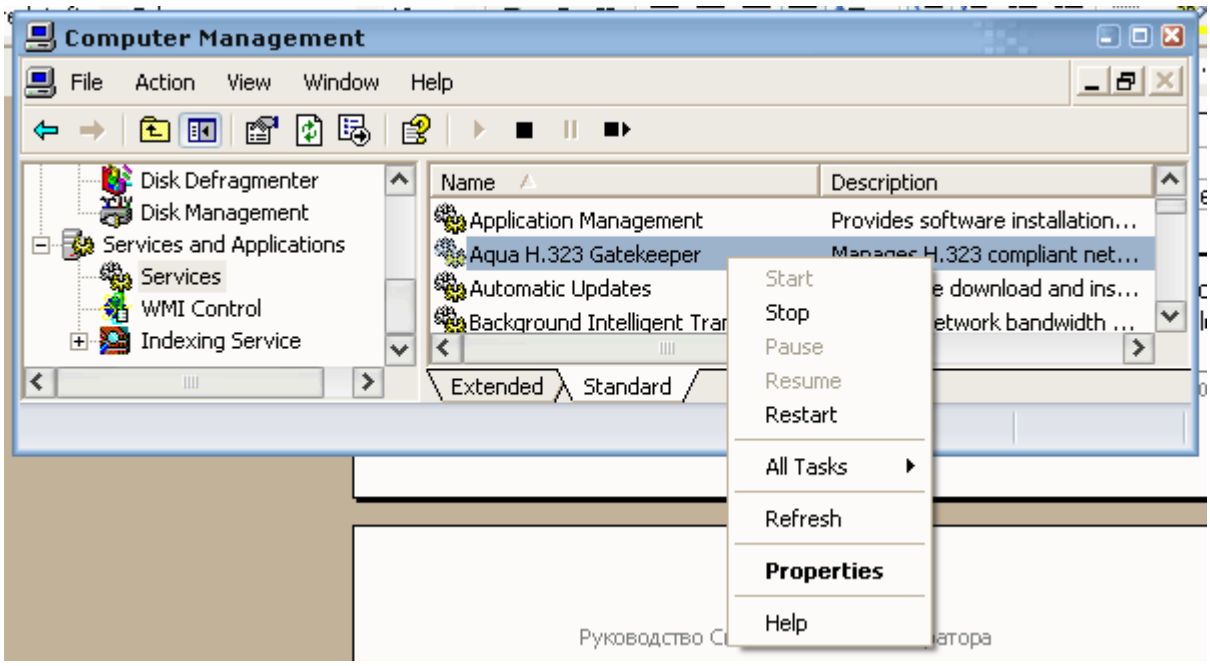
```
SGC.exe --conf=Config/SGC.cfg
```

Service

This is a primary mode of SGC functioning. It is convenient to use for the automatic start up when starting the operating system. In this mode SGC do not use any windows. At that, all log is written into the file specified using **log file <filename>** directive**;** all error-caused events that require special attention are recorded into the system log of the Operating System (EventLog).

When having the SGC in the "service" mode you have two options:

1. Using "Computer Manager" system utility choose Services item and in the context menu select Start.



2. Using command prompt execute the command:

```
SGC.exe --start
```

or

```
net start SGC
```

After that your system is ready to work and configuring.

IV. Configuration

This chapter is build in the following way: in the beginning there is a description of the system configuration and a subsidiary debugging information output system; afterwards a detailed description of the subscribers registration methods and callee subscriber selection mechanism are given; aliases translation and numeration plans coordination of different systems are also presented in this chapter. At last there is a description given of how to combine several SGCs in one network supplied with examples and options available.

Before you start reading this chapter it is strongly recommended to review all vocabulary and **basic concepts** used in this manual.

1. Basic configuration

The configuration file (supplied with the software) allows SGC to register any subscribers with any aliases. Registered subscriber will be available via aliases (numbers) which were provided for the registration. This configuration file is only an example and is not used for regular work as you do not have any opportunity to see what subscribers are being registered and where the calls will be terminated. This file just displays that SGC has a trivial functionality described in H.323 standard.

After subscriber's registration on the SGC you can start making calls. But at first, you should delete the string in the configuration which permits everyone to call everywhere.

```
/*
  user Anyone login "(.+)" set UserAlias add "\1",
               alias UserAlias;
*/
```

In the **system** section it is necessary to set up an IP-address which will be used by SGC. Otherwise, SGC will use the address which is specified in DNS of the computer, which is generally not a right thing to do, as the computer might have several interfaces and/or several IP-addresses on each interface (IP alias). In this case you take the control over the situation by specifying your own address. According to this methodology you can run several instances of SGC, each of them having its own IP-address.

In this section one should specify a gatekeeper identifier. This identifier is of a big importance for the subscribers' work using RAS protocol because SGC is checking the value in the RAS-packets transmitted by subscribers with that specified in the configuration.

Example

```
system
{
  identifier 'AGK';
  address 192.168.0.1;
}
```

It is also important to mention that configuration file syntax allows describing 'complex expressions' like **system**, **log**, **user**, **group** and others using two different methods. These methods are described in 'Configuration File Syntax' chapter' - **Expressions**

Example

```
system identifier 'AGK', address 192.168.0.1;
```

In order to control and monitor the system it is required to set up the debug information output subsystem and working protocol using **log** section, where the output **file** and information to be outputted is specified.

Example

```
log
{
  file '/home/gatekeeper/logs/gatekeeper.log';
  RAS messages yes;
  Q931 messages yes;
  H245 messages yes;
  radius messages yes;

  messages dump no;

  registrations          yes;
  connection status      yes;
  chosen route           yes;
  available bandwidth    yes;
  bandwidth changes      yes;
}
```

For all changes to take effect in the configuration, perform one of the following actions:

- Reload Configuration (SGC Monitor)
- Send a signal to USR1 processor using **kill** command (Unix)

The commands listed do not interrupt current connections if the users had not been deleted (**user**, **gatekeeper**); and the zones and links (**zone**, **link**) that take part in the connection and the taken bandwidth was not exceeded (**bandwidth**, **connections**). All new connections will work using new rules.

As it was mentioned above, the system monitoring, checking and changing of the configuration file is most conveniently accomplished using SGC Monitor. For SGC Monitor access limitations you can set up a password and/or limit the number of addresses from which the user can connect to it. The protocol of Monitor and Gatekeeper communication is an open source and its **description** is available in the **Supplementary Information chapter**.

2. Subscribers registration methods

Any call in SGC can be performed only between registered subscribers. If SGC could not identify the subscriber that wants to register, each request from this subscriber will be rejected.

SGC supports two types of subscribers' registration: RAS and static registration. Both of these types support a complete scheme of calls routing, data bandwidth control, hunting. However, RAS registration is more preferable because the usage of this protocol more flexibility and service speed is obtained.

RAS

RAS registration – this is a basic gatekeeper subscribers' registration type described in ITU-T H.323 standard.

For better understanding we will pay a bit more attention to the RAS registration procedure. The terminal (IP-phone, VoIP gateway, gatekeeper, other device or a program) that wishes to register on a gatekeeper must send a RegistrationRequest packet (RRQ) on SGC RAS UDP port (1719 port by default) specifying its Call Signal (CS) address, RAS address and the list of login aliases which it wants to be registered. The login aliases list is a key parameter because by its value SGC identifies the subscriber and finds it in the configuration; only after this procedure other checks are performed, i.e. IP-address checking (see **address** expression), H.235 information (see **h235_authentication** expression).

If the gatekeeper fails to find the subscriber with all specified login aliases or there are several subscribers with listed aliases, the RegistratioReject (RRJ) packet will be sent . If all the parameters match the gatekeeper approves the registration by sending the RegistrationConfirm packet (RCF) and assigns a unique identifier

(EndpointIdentifier) to the subscriber; the gatekeeper specifies the time value within which the subscriber must acknowledge its registration (TimeToLive, see **registration validity**). After that the subscriber is considered to be registered in the system and can initiate and receive calls.

When using this method of registration one must specify the list of aliases provided by the subscriber using '**login**' expressions. If you are not sure what aliases are provided for the registration you can turn on the RAS-messages output on (using '**log RAS messages**' expression) and view the RRQ message. Please pay attention to the TerminalAlias list – it contains the login aliases. Each of the aliases specified in this list should match with at least one of the **login** expression in the configuration of the subscriber (user).

Example

```
RAS, got message from 192.168.0.2:3954
Message RegistrationRequest {
  RequestSeqNum: 3 (0x3)
  ProtocolIdentifier: { 0, 0, 8, 2250, 0, 4 }
  DiscoveryComplete: true
  CallSignalAddress count(1) {
    #0 IpAddress {
      Ip: 4 octets [C0 A8 00 02] (192.168.0.2)
      Port: 1720 (0x6b8)
    }
  }
  RasAddress count(1) {
    #0 IpAddress {
      Ip: 4 octets [C0 A8 00 02] (192.186.0.2)
      Port: 3954 (0xf72)
    }
  }
  TerminalType {
    Vendor {
      Vendor {
        T35CountryCode: 181 (0xb5)
        T35Extension: 0 (0x0)
        ManufacturerCode: 0 (0x0)
      }
      ProductId: 'AquA IPhone'
      VersionId: '1.0.1'
    }
    Terminal {
    }
    Mc: false
    UndefinedNode: false
  }
  TerminalAlias count(2) {
    #0 DialedDigits: '1001'
    #1 H323_ID: 'john'
  }
  GatekeeperIdentifier: 'AGK'
  EndpointVendor {
    Vendor {
      T35CountryCode: 181 (0xb5)
      T35Extension: 0 (0x0)
      ManufacturerCode: 0 (0x0)
    }
    ProductId: 'AquA IPhone'
    VersionId: '1.0.1'
  }
  TimeToLive: 60 (0x3c)
  KeepAlive: false
  WillSupplyUUIEs: false
  MaintainConnection: false
```

```
  SupportsAltGK
}
```

The following table describes the key parameters in the packet:

Parameter	Value	Description
CallSignalAddress	192.168.0.2:1720	IP-address, that the device registers on the gatekeeper
TerminalAlias	1001, john	List of aliases using which the gatekeeper defines the subscriber
GatekeeperIdentifier	AGK	A Gatekeeper identifier to which the subscriber wants to connect; must match the **identifier** specified in the configuration

For the successful registration there should be a corresponding description of the subscriber in the configuration (**user**)

Example

```
user JohnDoe
{
  login 'john';
  login '1001';
  address 192.168.0.2 allow;
}
```

or you may not specify the **address** parameter. In this case the subscriber can be registered from any address.

Example

```
user JohnDoe
{
  login 'john|1001';
}
```

After the RRQ receiving and a successful registration completion the gatekeeper will send the RCF packet which specifies the EndpointIdentifier and TimeToLive parameters.

```
User registered:  JohnDoe at 192.168.0.2:1720
RAS, send message to 192.168.0.2:3954
Message RegistrationConfirm {
  RequestSeqNum: 3 (0x3)
  ProtocolIdentifier: { 0, 0, 8, 2250, 0, 4 }
  CallSignalAddress count(1) {
    #0 IpAddress {
      Ip: 4 octets [C0 A8 00 01] (192.168.0.1)
      Port: 1720 (0x6b8)
    }
  }
  TerminalAlias count(1) {
    #0 DialedDigits: '1001'
    #1 H323_ID: 'john'
  }
  EndpointIdentifier: '81ebc4'
  AlternateGatekeeper: <empty>
  TimeToLive: 60 (0x3c)
  WillRespondToIRR: true
  MaintainConnection: false
}
```

Now the subscriber can perform calls. For the calls receiving, using **alias** expression, you should set up a list of aliases which terminates the calls. For the detailed information see '**administrative aliases assigning**' chapter.

Example

```
user JohnDoe
{
  login 'john|1001';
  alias '1001';
}
```

You can call to JohnDoe subscriber by typing 1001 after its successful registration.

Static

Static registration does not imply a subscriber's registration itself as well as any other of RAS-procedures. The static subscriber will be registered in the system automatically after the system startup. The Static subscriber declaration is performed using '**static**' expression.

Example

```
user GW3
  static 192.168.1.1;
```

If the subscriber implies a call termination procedure, one should specify a list of aliases which are used to make this subscriber available like it was done for the RAS-subscriber. For the detailed description see '**administrative aliases assigning**' chapter.

Example

```
user GW3
{
  static 192.168.1.1;
  alias '22\d{4}';
}
```

You can make a call to GW3 subscriber right after the system startup by typing "22" plus any 4 digits.

The **address** expression has a different meaning for the static subscribers. This address and/or a range of addresses, the calls to which will match this subscriber using CS-port of the gatekeeper (Q.931:Setup packet).

The gatekeeper will reject the call received via its CS-address if it cannot correlate it with any of subscribers.

As the **static** expression automatically includes the **address** expression, before using **static** expression one should exclude the corresponding IP-address if there some more subscribers which calls may come from the same address.

Example

```
// Incoming calls only from SomeNetwork:192.168.100.0-255
user SomeNetwork
{
  address 192.168.100.0-255;
  static; // used for static registration
}

// Terminate calls only to 192.168.100.1
user SomeNetworkGateway
{
  address 192.168.100.1 false;
  static 192.168.100.1;
  alias '7095.+';
}
```

Or visa versa – exclude termination gateway address from the subscriber that receives calls from the specified range of address including the gateway address.

Example

```
// Incoming calls only from SomeNetwork:192.168.100.0-255
user SomeNetwork
{
  address 192.168.100.1 false; // SomeNetworkGateway's IP
  address 192.168.100.0-255;
  static; // used for static registration
}

// Receive and Terminate calls at 192.168.100.1
user SomeNetworkGateway
{
  static 192.168.100.1;
  alias '7095.+';
}
```

The «**address** 192.168.100.1 **false**;» expression is necessary. As it was mentioned before, the «**static** 192.168.100.1;» expression implies «**address** 192.168.100.1;» expression. Thus, when having a call from 192.168.100.1 IP-address SGC will have two subscribers: «SomeNetwork» and «SomeNetworkGateway», which own this IP-address; this call will be rejected because the gatekeeper requires only one caller subscriber for the billing. This ambiguity is resolved by «**address** 192.168.100.1 **false**;» expression.

RAS advantages

Subscribers' registration using RAS-protocol has a set of advantages comparing to the static registration:

- Subscriber availability control (for example, the subscriber is not registered - turned off, broken or not connected). In this case, SGC will not waste its time attempting to call to an unexciting device and this lets the gatekeeper to switch to the alternative route and to decrease the connection establishment time and/or to give a negative answer to the caller subscriber (see **registration validity**)
- Taken bandwidth precise control. The RAS-protocol implies a procedure according to which the subscriber indicates the actual taken bandwidth in the channel. This feature provides a VoIP network with planning/control/management features (see **zone bandwidth**, **link bandwidth**)
- Established/buzzed connections control. Using the RAS-protocol the gatekeeper has an opportunity to force the subscriber to send special InfoRequesResponce (IRR) packets in the time period specified by the administrator during all call durability. The absence of these packets means that the connection is not active and the gatekeeper will finalize it. (see **connection validity**)
- The RAS-protocol has a feature of alternate gatekeepers usage. This feature provides with an opportunity for the load distribution management for the specific gatekeepers and their exchangeability in case of failures. (see **alternate gatekeeper**)
- For the RAS subscribers one can perform their accounting and registrations authentication in the billing system even before they attempt to make a call (see **radius accounting login**)
- During the registration, the RAS subscribers can create variables which values can be used later during their work. For example, this value can act as a RADIUS-server password. Using a login alias one can create aliases which will make the subscriber available. (see **make login alias**)

3. Administrative aliases assigning

SGC has a feature to assign any aliases to the subscriber, no matter what aliases are registered by the subscriber. The '**alias**' directive is used for this purpose.

Use '**login**' command in user's record to specify all aliases which it registers. These aliases are used for the subscriber correlation with the user's record (see **RAS subscribers registration**). A registered subscriber will not be available via those aliases. In order to specify aliases that are used for the subscriber's attainability one must list them using '**alias**' directive.

Example:

```
user UserOne
{
  login 'login1';
  alias '101';
}

user UserTwo
{
  login 'login2';
  alias '102';
}
```

In this example the subscriber which is registered under 'login1' alias will be attainable using '101' alias; the subscriber which is registered under 'login2' alias will be attainable using '102' alias.

All above information also corresponds with subscribers that have static registration.

Example:

```
user UserOne
{
  static 192.168.0.10;
  alias '101';
}

user UserTwo
{
  static 192.168.0.11;
  alias '102';
}
```

When the subscriber should terminate calls for several numbers (i.e. 4-port FXS gateway) one can use several **alias** expressions when describing the subscriber.

Example:

```
user fxs
{
  login 'fxs';
  alias '101';
  alias '102';
  alias '103';
  alias '104';
}
```

SGC terminates the call on the registered subscriber "fxs" if the **callee alias** has one of the following values: 101, 102, 103, 104.

As it is done in the **alias** expression, the alias can be specified using «**regular expression**». Thus, there are several ways of describing this subscriber.

Example:

```
user fxs
{
  login 'fxs';
  alias '101|102|103|104';
}
```

or


```
user fxs
{
  login 'fxs';
  alias '10[1-4]';
}
```

In such a way using «**regular expressions**» one can describe the direction using one template, if it is required for yours and provider's VoIP gateways description.

Example

```
user gateway1
{
  static 1.1.1.1;

  alias '7095.+';          // Moscow
  alias '7343212.+' false; // exclude Ekaterinburg 212
  alias '7343.+';          // Ekaterinbug
}

user gateway2
{
  static 1.1.1.2;
  alias '7343212.+';
}
```

Note *This example not only shows a possibility of using **regular expressions** for the direction description but it shows the way of using alias <bool> expression which performs direction exclusion.*

The order in which the lines are placed in the record is important because SGC stop on the first matching alias.

Right way:

Example:

```
alias '7343212.+' false; // exclude Ekaterinburg 212
alias '7343.+'; // Ekaterinbug
```

Wrong way:

Example:

```
alias '7343.+'; // Ekaterinbug
alias '7343212.+' false; // exclude Ekaterinburg 212
```

«**alias** '7343212.+' **false**;» is not processed because «**alias** '7343.+';» implies «7343212.+».

4. Subscribers authentication calls accounting on RADIUS

SGC has a feature of authentication and accounting directly in your billing system using RADIUS protocol.

RADIUS parameters configuration

RADIUS-server parameter must be specified in **aaa** section. The most significant parameters are **address** and **password**. If your RADIUS-server uses standard ports UDP:1812 and UDP:1813, their configuration can be skipped; they will be used by default. One can specify a time zone value which is used by SGC accounting packets; the following expressions are used: **time zone name** and **time zone offset**. There is an important feature provided, that allows accounting for not only successful calls but failed calls either; in order to do that use **log failed calls** expression.

To debug the information which is sent between SGC-NAS and RADIUS-server use «**log radius messages true**» expression; the packets which are shown below in this chapter are acquired by means of this expression.

Example

```
aaa
{
  address 192.168.1.1;
  authentication port 1812;
  accounting port 1813;
  log failed calls true;
  time zone name 'YEKST';
  time zone offset local;
}
```

After that, for each subscriber should be specified what is to be done. **Radius authentication** and **radius accounting** are configured separately.

Authentication

There are several modes for subscriber authentication: **registration**, **call initiating**, **call answering**.

Registration

For the RAS-subscribers the Access-Request is sent with Service-Type attribute value of "Login".

This is configured by **radius authentication** expression in login mode.

Example

```
user abonent
{
  login 'abonent';
  radius name 'user';
  radius password 'password';
  radius authentication login;
}
```

After receiving RegistrationRequest (RRQ) packet and login aliases matching with the subscriber ("user abonent") (see **RAS subscribers registration**), SGC sends Access-Request packet to the RADIUS-server.

Example

```
RADIUS, send message to 192.168.1.1:1812
{
  Code: Access-Request
  Identifier: 1
  Length : 79
  Authenticator: FC 79 00 00 39 74 00 00 F5 77 00 00 18 0F
  Attributes {
    NAS-IP-Address: 192.168.0.1
    NAS-Identifier: "AGK"
    User-Name: "user"
    User-Password: 29 97 18 85 F6 7B A6 3C B9 CB D3 3C 81
    Service-Type: Login
  }
}
```

Only after receiving the Access-Accept packet from the RADIUS-server the system permits subscriber's registration by sending RegistrationConfirm packet. In any other case the registration will be refused.

Call initiating

Call initiating authentication is performed both for RAS-subscribers and static subscribers; the Access-Request is sent with Service-Type attribute having "Call Check" value.

This is configured by **radius authentication** expression in the "caller" mode.

Example

```
user abonent
{
  login 'abonent';
  radius name 'user';
  radius password 'password';
  radius authentication caller;
}
```

After receiving AdmissionRequest (ARQ) packet from the RAS-subscriber or Q931:Setup from the static subscriber, SGC sends Access-Request to the RADIUS-server; the system permits the call only after receiving a positive reply from RADIUS. In the reply Access-Accept packet the RADIUS-server can specify the maximum call durability in Session-Timeout and/or Cisco VSA h323-credit-time attributes. SGC interrupts the call after timeout expiration.

Example

```
RADIUS, send message to 192.168.1.1:1812
{
  Code: Access-Request
  Identifier: 2
  Length : 304
  Authenticator: 8E700000A72C0000803E00003D1F0000
  Attributes {
    NAS-IP-Address: 192.168.0.1
    NAS-Identifier: "AGK"
    User-Name: "user"
    User-Password: 2D221478A5A10B8C71A772A9E13E814F
    Service-Type: Call check
    Calling-StationId: "abonent"
    Called-StationId: "1010"
    Vendor-Specific 9 Cisco: 26 "h323-call-origin=answer"
    Vendor-Specific 9 Cisco: 27 "h323-call-type=VoIP"
    Vendor-Specific 9 Cisco: 24 "h323-conf-id=C0F754BE..."
    Vendor-Specific 9 Cisco: 23 "h323-remote-
address=192.168.1.10"
    Vendor-Specific 9 Cisco: 25 "h323-setup-
time=07:33:00.364 UTC Mon May 31 2004"
  }
}

RADIUS, got message from 192.168.1.1:1812
{
  Code: Access-Accept
  Identifier: 2
  Length : 108
  Authenticator: 9D697A885FC619262B02E05108878027
  Attributes {
    Vendor-Specific 9 Cisco: 102 "h323-credit-time=3600"
    Vendor-Specific 9 Cisco: 101 "h323-credit-
amount=400.00"
    Vendor-Specific 9 Cisco: 103 "h323-return-code=0"
  }
}
```

Call answering

SGC has a feature of permitting requests from RADIUS not only for the caller but for the callees. This can be extremely useful if your billing system supports this mode.

This is configured by **radius authentication** in the «callee» mode.

Example

```
user abonent
{
  login 'abonent';
  radius name 'user';
  radius password 'password';
  radius authentication callee;
}
```

The principle of functioning and the attributes in the packets are the same as in the case of a **call initiating** with only one difference: Cisco VSA h323-call-origin has a value of "originate".

Username and password

For all types of RADIUS authentication one must specify username and password. These data is required for the correct functioning. These parameters are configured using the following expressions: **radius name**, **radius password**.

The following identifiers can be used as parameters:

- Character strings
- Variable names
- H.235 subscriber information (saved in special variables)

Several subscribers

Using variable names which values are taken from the registration information (aliases registration) and/or from the call information (aliases of the caller and/or the callee) can be described using one "user" expression for several subscribers which is very convenient and flexible when having up-to-date billing system.

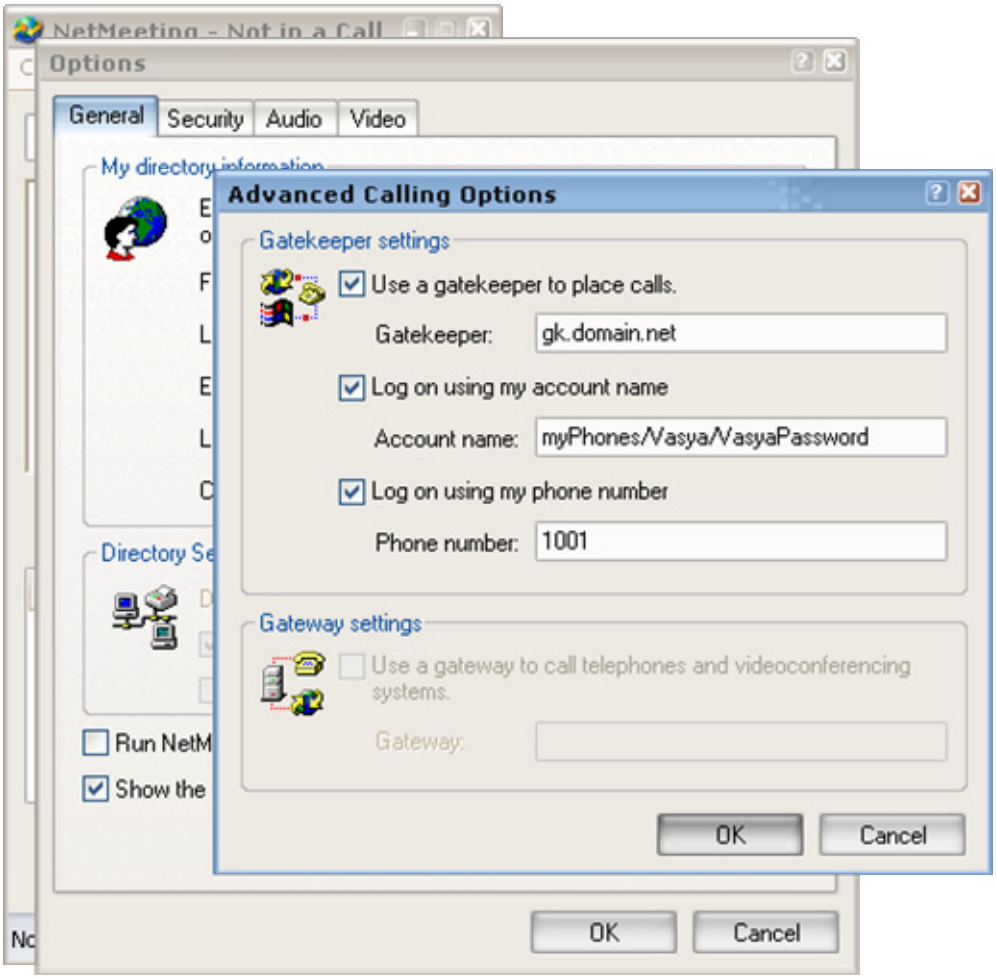
Example

```
user MyPhones
{
  login 'myPhones.+|\d{4}';
  make login 'myPhones/(.+)/(.+)' set MyName to '\1'
                                   set MyPass to '\2';

  make login '(\d{4})' set MyAlias add '\1';
  alias MyAlias;
  radius name MyName;
  radius password MyPass;
  radius authentication login|caller;
  radius accounting call;
}
```

The above example shows how the information for the RADIUS is extracted from "login alias" and how aliases are determined; at that all registration procedures are performed by your billing system. You only have to set up the subscriber properly.

Example



Accounting

Calls accounting is performed on the RADIUS-server and is configured by **radius accounting** expression and is set up for each subscriber separately.

Every call for SGC consists of two "call legs":

- Answer – incoming call from the caller; in RADIUS packets it is marked using Cisco VSA «h323-call-origin=answer»
- Originate – outgoing call to the callee; in RADIUS packets it is marked using Cisco VSA «h323-call-origin=originate»

Thus, there is an option to enable/disable the accounting information sending for each call "leg" separately by specifying the mode of radius accounting:

- caller – answer call leg
- callee – originate call leg
- call – answer and originate call legs

Example

```
user callerAbonent
{
  login 'Abonent1';
  radius accounting caller;
}

user calleeAbonent
{
  login 'Abonent2';
  radius accounting callee;
}
```

In this example, when a call from "callerAbonent" to "calleeAbonent" occurs, SGC sends four packets "accounting Start/Stop" for each call "leg". If the call is performed in an opposite direction (from "calleeAbonent" to "callerAbonent"), no packets will be sent to the RADIUS-server. In order to send all complete accounting information both of these cases, each subscriber that take part in the call should be configured using «**radius accounting call**» expression.

Example

```
user Abonent1
{
  login 'Abonent1';
  radius accounting call;
}

user Abonent2
{
  login 'Abonent2';
  radius accounting call;
}
```

The most common thing to do is to specify «**radius accounting call**» expression in "everyone" group; in this case the accounting is performed for all subscribers and there is no further need to specify accounting parameters for each subscriber separately.

Example

```
group everyone
  radius accounting call;

user Abonent1 login 'Abonent1';
user Abonent2 login 'Abonent2';
```

5. Alias translation

Aliases translation, which is one of the most important functional features of SGC, is generally used in:

- numeration plan coordination
- specific directions blocking
- subscriber dialing simplification
- multi-channel numbers configuration
- useful information extraction, i.e. for RADIUS-server

Translation methods

SGC has a feature of processing both **callee alias** and **caller alias**. The translation can be performed both on the input when getting the alias from the subscriber (outgoing) and on the output when sending aliases to the subscriber (incoming).

To configure aliases translation use «**translate ... alias ... to**» expression.

Note. The direction is specified from the subscriber's side

- outgoing – aliases are sent from the subscriber to SGC;
- incoming – aliases are sent from SGC to the subscriber.

Numeration plan coordination

This chapter describes the numeration plan coordination of IP-telephony providers, to which your VoIP-network is connected, with inner numeration plan of your system and billing.

Usually, there is a common numeration plan used in the system; that means that for each termination point and for the points outside the system, there is a

unique alias which has some restrictions for its formation. By this, the exclusion of intersecting aliases from different networks is obtained.

Your inner numeration plan lets you to determine which devices terminate the traffic; and the aliases in the inner numeration plan are sent to the billing system.

Imagine that you have a transit traffic that is sent from provider A to provider B and visa versa, some part of this traffic is terminated in your city on your gateway and you have some IP-subscribers.

Your inner numeration plan is configured for Russia and has the following format:

7<city code><number>, 11 digits total

Termination point	Description

ProviderA	The subscriber – VoIP gateway of "Provider A", that performs calls termination for Moscow, accepts dialed numbers in 7-digit format (Moscow format), you receive calls from this subscriber in the following format: 8+ <city code><phone number>.
	The alias translation of a callee is required both when sending him a number (cut 7095):
	`translate incoming callee`
	`  alias '7095(\d{7})' to '\1';`
	and when receiving a number ('7' instead of '8'):
	`translate outgoing callee`
	`  alias '8(\d{10})' to '7\1';`

ProviderB	The subscriber – VoIP gateway of "Provider B", that peforms calls termination thoughout all Russia including Moscow, but with hiher pricing than «Provider A". That is why different **cost** parameters are set for this direction.
	The gateway receives and transmits numbers in the following 10-digit format:
	<city code> + <phone number>
	It is necessary to cut '7' from the number sent to this gateway:
	`translate incoming callee`
	`  alias '7(\d{10})' to '\1';`
	and add '7' when receiving the call from it:
	`translate outgoing callee`
	`  alias '(\d{10})' to '7\1';`

Gateway	This is your personal gateway that terminates calls for your city (e.g. Ekaterinburg). The set up is similar to "Provider A" with the only difference that the city code is '343':
	`translate outgoing callee`
	`  alias '8(\d{10})' to '7\1';`
	`translate incoming callee`
	`  alias '7343(\d{7})' to '\1';`

AbonentNº	IP-phones/gateways of your subscribers. For them, there is a possibility of dialing in the following formats:
	• 7 digits of Ekaterinburg number
	• 10 digits to call to another city
	• 8 + (10 digits to call to another city)
	`translate outgoing callee`
	`  alias '(\d{7})' to '7343\1';`
	`translate outgoing callee`
	`  alias '(\d{10})' to '7\1';`
	`translate outgoing callee`


```
  alias '8(\d{10})' to '7\1';
```

7-digit number will be received by the subscriber that allows specifying a 7-digit number on the port of the terminal device.

```
translate incoming callee
  alias '7343(\d{7})' to '\1';
```

Not to describe similar aliases translation for each subscriber we use "Ektb" group.

Example

```
// --- Accounting for all subscribers----------------------
group everyone
{
  radius accounting call;
  dial Self deny; // Restrict calling to themselves
  dial allow; // Allow using all others
 }

// --- Moscow termination ----------------------------
user ProviderA
{
  static 192.168.10.1;

  alias '7095\d{7}' cost 10;
  translate outgoing callee alias '8(\d{10})' to '7\1';
  translate incoming callee alias '7095(\d{7})' to '\1';
}

// --- Russia termination ----------------------------
user ProviderB
{
  static 192.168.20.1;

  alias '7343\d{7}' false; //only «user Gateway» serves
"343" direction
  alias '7095\d{7}' cost 100;
  alias '7\d{10}';
  translate outgoing callee alias '(\d{10})' to '7\1';
  translate incoming callee alias '7(\d{10})' to '\1';
}

// --- local gateway in Ekaterinburg ----------------------
user Gateway
{
  login 'myGateway';


  alias '7343\d{7}';
  translate outgoing callee alias '(\d{7})' to '7343\1';
  translate outgoing callee alias '8(\d{10})' to '7\1';
  translate incoming callee alias '7343(\d{7})' to '\1';
}

// --- Local subscribers translation -----------------
group Ektb
{
  translate outgoing callee alias '(\d{7})' to '7343\1';
  translate outgoing callee alias '(\d{10})' to '7\1';
  translate outgoing callee alias '8(\d{10})' to '7\1';

  translate incoming callee alias '7343(\d{7})' to '\1';
}
```

```
// --- Subscribers-----------------------------------
user Abonent1
{
  login 'abonent1';
  alias '73431111111';
  group member Ektb;
}

user Abonent2_2FXS
{
  login 'abonent2|222222[1-2]';
  alias '7343222222[1-2]'; // local:2222221, 2222222
  dial Self allow; // allow calling from port to port
  group member Ektb;
}
```

Specific directions blocking

Aliases translation also can be used to block some specific numbers and/or directions; if the alias is translated into an empty string the call is not going anywhere.

For example we have a subscriber that is allowed to make only local calls and calls to Moscow. To block all other directions, it is necessary to translate all other aliases into empty ones. Actually we could have used «**dial** ProviderB **deny**;» expression, but in this case the subscriber would loose a possibility to call to Moscow – this is not acceptable for us.

Example

```
user Abonent1
{
  login 'abonent1';
  alias '73431111111';
  translate outgoing callee alias '\d{7}' to '7343\0';
  translate outgoing callee alias '095\d{7}' to '7\0';
  translate outgoing callee alias '8095\d{7}' to '7095\0';
  translate outgoing callee alias '.+' to '';
  group member Ektb;
}
```

Note. *We had to re-repeat aliases translation (from subscribers) accepted in "Ektb" group, because SGC stops on the first matching "regular expression"; but we managed to inherit aliases translation (to the subscribers) from the group.*

If the device has more than one port, for example «Abonent2_2FXS», there is a unique feature to block the specific direction for one port only and/or different directions for a set of ports. Using caller aliases, calling from the first port, it is "2222221", from the second one – "2222222". We will limit the second port.

Example

```
user Abonent2_2FXS
{
  login 'abonent2|222222[1-2]';
  alias '7343222222[1-2]'; // local:2222221, 2222222

  // variables preparation for the first port
  make outgoing caller alias '2222221'
    set LocalNumber to '7343\\1'
    set MSKNumber to '7\\1'
    set Russia to '7\\1';

  // variables preparation for the second port
  make outgoing caller alias '2222222'
    set LocalNumber to '7343\\1'
```



```
    set MSKNumber to '7\\1'
    set Russia to '';

  translate outgoing callee
    alias '(\d{7})' to LocalNumber;
  translate outgoing callee
    alias '(095\d{7})' to MSKNumber;
  translate outgoing callee
    alias '810(.+)' to Russia;

  dial Self allow; // allow calling from port to port
  group member Ektb;
}
```

Multi-channel numbers configuration

Despite of the fact that the device awaits for the specific number to redirect the call to the specific port, using aliases translation you can translate the aliases into the awaited one.

Example

```
user abonent1
{
  login 'ab1';
  alias '1001'; // awaited alias
  alias '000'; // common with abonent2

  translate incoming callee alias '.+' to '1001';
}

user abonent2
{
  login 'ab2';
  alias '1002'; // awaited alias
  alias '000'; // common with abonent1

  translate incoming callee alias '.+' to '1002';
}
```

Useful information extracting

The information that is sent by the subscriber in callee alias and/or caller alias can be sent to the RADIUS-server card platform.

Let's imagine, that "gw1" subscriber must add three digits of card number and three digits of PIN-code before the number he wants to dial; this feature lets to organize card platform on the gateways without IVR support.

Example

```
user gw1
{
  static 192.168.0.2;

  make outgoing callee alias '(\d{3})(\d{3})(.+)'
    set Card to '\1'
    set PIN to '\2'
    set CalleeAlias to '\3';

  translate outgoing callee alias '.+' to CalleeAlias;

  radius name Card;
  radius password PIN;
  radius authentication caller;
}
```

6. Traffic proxy

The gatekeeper has a feature of signaling traffic and media data traffic proxy. However, proxy can be completely disabled. By default, the gatekeeper has a feature of signaling traffic proxy. Proxy level is configured using «**proxy level**» expression. For the connection the highest among two levels for the callee and the caller proxy level is selected.

If you need to perform connection duration precise accounting, it is not recommended to use the gatekeeper with proxy mode disabled, as the connection duration information in this case may not be precise. To do that a signaling traffic proxy mode is enough to be used.

Full proxy «proxy level full» allows both controlling signalization going through the gatekeeper and depersonizing the traffic for your provider; also it lets to improve your network's security and use "fake" IP-addresses for your subscribers.

Generally, full proxy can be enabled for the outer world connection and this provides with optimal security, performance and precise accounting.

Example

```
system identifier 'SGC', address <real IP>;

group external
  proxy level full;

user ExternalGW
{
  static <real IP>;
  group member external;
}

user abonent1
{
  static 192.168.0.1;
}

user abonent1
{
  static 192.168.0.2;
}
```

In this example the local subscribers have "fake" IP-addresses and do not have an access to the real ones (except for the SGC address); and voice traffic goes straight between the subscribers in case if the full proxy is turned on for the "ExternalGW" connection.

7. Termination point selection

There can be several subscribers in the system with the same aliases. In this case the subscribers go through the balancing procedure in order to determine the only terminating subscriber.

Example

```
user gw1
{
  static 192.168.0.10;
  alias '7095.+';
  alias '7343.+';
}

user gw2
{
```



```
  static 192.168.0.11;
  alias '7095.+';
  alias '790284.+';
}
```

In this example when having a call that starts with "7095", SGC has two options of where to terminate the call: «gw1», «gw2». Termination point selection is performed according to the **balancing procedure**.

Balancing

Subscribers are balanced according to the following scheme:

- The subscriber with less cost wins.
- If costs are equal, the subscriber with wider bandwidth wins.
- If the bandwidth is equal for both subscribers, the subscriber with less current load wins.
- If the current load is equal, the subscriber is chosen randomly.

Note. Subscriber load is calculated using this formula: Load = Current Connections Count+1/Max Connections Count.

«Max Connections Count» is set in «**connections**» expression of the subscriber description.

The cost can be specified for the subscriber using «**cost**» expression or separately for each «**alias**».

Example

```
user abonent1
{
  alias '7095.+' cost 1;
  alias '7343.+';
  alias '1000';
  cost 10;
}

user abonent2
{
  alias '7095.+';
  alias '7343.+' cost 1;
  alias '1000';
  cost 100;
}
```

When a call which number starts with "7095" arrives, it goes to "abonent1" subscriber; if the number starts with "7343" – it goes to "abonent2" subscriber; if the alias is "1000", the call is directed to the "abonent1" subscriber.

Current available bandwidth is defined as a subtraction of the available bandwidth which is specified by «**bandwidth**» expression in «**zone**» and «**link**» between the zones where the subscribers are located and actual bandwidth which is taken by all the connections that are established in these zones. The bandwidth taken by one connection for RAS-subscribers is extracted from the information specified by the subscribers in RAS-packets: AdmissionRequest (ARQ), BandwidthRequest (BRQ) of connection establishing (bandwidth field); for the static subscribers the bandwidth taken for one connection is specified by the «**bandwidth**» expression in subscriber's description «**user**».

Example

```
zone A bandwidth 1G;
zone B bandwidth 500K;
```

```
user A1
{
  location A;
  login 'AUser';
}

user A2
{
  location A;
  static 192.168.0.2;
  bandwidth 16K;
  alias '1000';
}

user B1
{
  location B;
  static 192.168.1.1;
  alias '1000';
}
```

When the "A1" subscriber calls "1000" number, "A2" subscriber is rather be chosen than "B1" subscriber, as B zone has much less specified bandwidth; however, it all depends upon the current load – actual taken bandwidth. The actual taken bandwidth and the bandwidth taken by one connection can be observed in the system log and/or in the Monitor.

8. Hunting

For the caller subscriber a sequential search of callee subscribers can be enabled. The search is executed according to the **balancing procedure** configuration until the match is found or the search list is empty. Search level is configured using «**hunt level**» command.

Search levels

SGC has several modes of the alternative subscribers searching:

- **none** – no search is executed in this mode. If the call cannot be accomplished at the moment (remote subscriber is busy or remote equipment is malfunctioning etc) the caller is not switched to the alternative subscriber and hears an error signal.
- **neutral** – the caller is switched to the alternative route in case of errors during the connection. However, if by this time the data channels are already established, the switch is not performed. The gatekeeper is not postponing the data channels establishing, and that is why this mode is called "neutral".
- **soft** – in this mode the Gatekeeper is trying to postpone the data channels establishing by moving the FastStart and H245Address information in Alerting message; this lets the subscribers to hear the RingBack tone if it is transmitted in the voice channel.
- **hard** – in this mode the Gatekeeper is trying to postpone the data channels establishing by moving FastStart and H245Address information in Connect message.

Note. *Note that in any of the modes the subscriber has an option of data channels establishing using H.245 tunneling thus making the further search impossible.*

The following example shows how to use the search efficiently (an example of VoIP stock exchange "Tario"). The specifics of stock exchange call processing implies several connection attempts for the successful call termination.

Example

```
group voipexch_gr
{
  proxy level full;

  address  212.53.35.0-255 false;
  static 212.53.35.34;

  h245 tunneling true;
  fast start true;

  // MSK
  alias "7095.+";
  alias "8095.+";
  alias "7499.+";
  alias "8499.+";

  // Smolensk
  alias "70812.+";
  alias "80812.+";

  translate incoming callee alias "710(.+)" to "\1";
  translate incoming callee alias "810(.+)" to "\1";
  translate incoming callee alias "8(.+)" to "7\1";
}

// all incoming from Tario
user voipexch
  address 212.53.35.0-255 true,
  group member voipexch_gr;

user voipexch2 group member voipexch_gr;
user voipexch3 group member voipexch_gr;
user voipexch4 group member voipexch_gr;
user voipexch5 group member voipexch_gr;
user voipexch6 group member voipexch_gr;
```

In this example up to 6 attempts are made to terminate the call on «**static** 212.53.35.34».

9. Gatekeepers interconnection

Several gatekeepers can be connected in one network. In the example below the gatekeepers are being registered at one another and are sending each other requests for aliases permission. In case of the successful alias translation the call is routed to the remote gatekeeper.

There are several ways of connecting SGC with other gatekeepers as well as a few ways of other gatekeepers' registration on the SGC; the behavior depends upon registration method. For the registration methods description the «**gatekeeper**» expression is used.

The «**gatekeeper**» expression implies all functionality and meaning of the «**user**» directive and this expands SGC features for working with H.323 terminals of the Gatekeeper type.

Types of registration

There are three types of registration

- With RAS-registration
- Without RAS-registration
- Static registration

RAS-registration

Using this method of registration, SGC works using the full scheme of RAS-protocol; the SGC behavior from the remote gatekeeper point of view will look like usual RAS-subscriber behavior:

- While registration SGC sends a RegistrationRequest packet; in case of the successful registration the following field values are supported: TimeToLive (for registration acknowledgement) and AlternateGatekeeper
- To make a call SGC sends a packet LocationRequest and as soon as it gets the acknowledgement it sends AdmissionRequest after receiving AdmissionConfirm, SGC follows all gatekeeper's instructions: bandwidth allocation - Bandwidth request and active connection acknowledgement InfoRequestResponse.
- After call completion a DisengageRequest packet is sent
- While shutdown, SGC disconnects from the gatekeeper using UnregistrationRequest packet.

This registration method has a feature of using password authentication via H.235 protocol.

Example

```
gatekeeper RemoteGK
{
  identifier 'remote@domain.net';
  ras address 192.168.1.1;
  register alias 'SGC';

  static;
  alias '.+';
}
```

Or perform the same procedure using H.235 with specifying the registration period value. Also in this example permission is given to use the list of alternative gatekeepers sent from "RemoteGK".

Example

```
system identifier 'SGC', address 192.168.0.1;

gatekeeper RemoteGK
{
  identifier 'remote@domain.net';
  ras address 192.168.1.1;
  register alias 'SGC';
  regitsration period 120;
  link H235 name 'SGC';
  link H235 password 'xxx';
  link H235 authentication true;
  link alternate gatekeeper true;

  static;
  alias '.+';
}
```

Without RAS registration

When using remote gatekeeper with no RAS-registration it is sufficient to specify RAS address and an identifier.

In this case SGC will register; when the call occurs the LocationRequest packet is sent first and Q.931:Setup is sent after acknowledgement. Anyway, this confirmation method has some advantages comparing to configuring the remote gatekeeper as a static user, because when having several alternatives the

LocationRequest packets are sent to several gatekeepers simultaneously; when getting first positive confirmation, Setup packet is sent – this is much faster than sending the Setup packet consequently to each of the subscribers.

Example

```
system identifier 'SGC', address 192.168.0.1;

gatekeeper RemoteGK
{
  identifier 'remote@domain.net';
  ras address 192.168.1.1;

  static;
  alias '.+';
}
```

Static subscriber

This way of the subscriber description was shown above in this manual; remote gatekeeper is described as an ordinary static subscriber and the communication with him is performed in the same way. Remote gatekeeper should support static subscribers.

Example

```
system identifier 'SGC', address 192.168.0.1;

user RemoteGK
{
  static 192.168.1.1;
  alias '7095.+';
}
```

Gatekeeper registration methods

It is mentioned above that «**gatekeeper**» expression is an expansion of the «**user**» expression; thus, gatekeeper registration on SGC has the same features as for the usual subscriber. See «**Subscribers registration methods**». Below are the examples for each method.

RAS

This description method corresponds with **RAS registration** with H.235 authentication.

Example

```
system
  identifier 'remote@domain.net',
  address 192.168.1.1;

gatekeeper SGC
{
  login 'SGC';
  registration validity 120;
  connection validity 60;

  h235 name 'SGC';
  h235 password 'xxx';
  h235 authorization true;
}
```

No registration, static

Corresponds with «**Without RAS-registration**»

Example

```
system
  identifier 'remote@domain.net',
  address 192.168.1.1;

gatekeeper SGC
{
  ras address 192.168.0.1;
  static 192.168.0.1;
}
```

Static subscriber

This method corresponds with «**Static subscriber**» registration.

Example

```
system
  identifier 'remote@domain.net',
  address 192.168.1.1;

user SGC
{
  static 192.168.0.1
}
```

Two SGC registration on each other

In this chapter we will sum up the information for gatekeeper interconnection. In the example it is shown a complete configuration for two SGC and their subscribers; their interconnection is performed using RAS-protocol and H.235 authentication. One of gatekeepers is serving Moscow direction and office IP-phones; the second one is located in Ekaterinburg and provides an access to IP-subscribers of the branch office and city numbers connection.

First SGC configuration, Moscow

```
system
{
  identifier 'MSK';
  address 192.168.0.1;
}

gatekeeper EKTB
{
  identifier 'EKTB';
  ras address 192.168.1.1;
  register alias 'MSK';

  link H235 name 'MSK';
  link H235 password 'MSK';
  link H235 authentication true;

  login 'EKTB';
  registration validity 120;
  connection validity 60;

  h235 name 'EKTB';
  h235 password 'EKTB';
  h235 authorization true;

  alias '.+';
  dial self deny;
  dial yes;
}
```



```
user moskow
{
  static 192.168.0.2;
  alias '7095.+';
}

user abonent1
{
  login 'abonent1';
  alias '1001';
}

user abonent2
{
  login 'abonent2';
  alias '1002
}
```

Second SGC configuration, Ekaterinburg branch office

```
system
{
  identifier 'EKTB';
  address 192.168.1.1;
}

gatekeeper MSK
{
  identifier 'MSK';
  ras address 192.168.0.1;
  register alias 'EKTB';

  link H235 name 'EKTB';
  link H235 password 'EKTB';
  link H235 authentication true;

  login 'MSK';
  registration validity 120;
  connection validity 60;

  h235 name 'MSK';
  h235 password 'MSK';
  h235 authorization true;

  alias '.+';
  dial self deny;
  dial yes;
}

user ekaterinburg
{
  static 192.168.0.2;
  alias '7095.+';
}

user abonent1
{
  login 'abonent1';
  alias '1003
}

user abonent2
{
```

```
    login 'abonent2';
    alias '1004;
  }
```

V.Configuration file syntax

Configuration file consists of a set of expressions. There are no limitations for characters positions and "space" character usage involved. A text, marked as a comment, is not interpreted as useful information.

1. Comments

Two types of comments are used – block and string. Block comment starts with '/*' characters and ends with '*/'. String comment starts with '//' characters and ends with the current line.

Example

```
/* This is a comment */
// This is also a comment
```

2. Expressions

Each expression starts with a reserved word which specifies the expression type. There are two types of expressions: simple and complex. Simple expressions perform a control over one property.

Example

```
login 'copah';
```

Complex expressions are much alike with simple ones. They include one or several simple expressions put in braces. In complex expressions, simple compounds are separated with commas and follow the complex expression declaration.

Example

```
user John
{
  location LAN;
  login "John";
  alias "John";
}

user Anyone
  group member Basic,
  login '(.+)' set UN to '\1',alias UN;
```

The order of expressions has a very important meaning. If there are several expressions of one certain type that control one property, only the first one is interpreted as a working one. Some expressions can be limited by their lifetime. These expressions have an optional reserved word '**at**' (See sub-expression '**at**').

Example

```
group Group1
{
  proxy level none;
  registration validity 70;
}

user User1
{
  registration validity unlimited;
  group member Group1;
  proxy level signalling;
  connections 4 at 9:00-18:00;
  connections 2 at 13:00-14:00;
  connections 1;
}
```

In this example the time of subscriber registration is not limited because the first expression of this type declares this state. The registration limitation which is described in the group has no power. However, '**proxy level**' expression in the user's configuration has no power because group declaration cancels this expression.

From 9:00 to 18:00 the subscriber has a possibility to make up to 4 connections; in other time – only one. The second rule (connections) has no power because its action time overrides the first expression having a higher priority.

3. Reserved words

Reserved words are not case-sensitive. The administrator can use any case in reserved words or even mix them. For example, '**file**' and '**File**' – it is the same reserved word. Most of the reserved words are used to specify the expression type and its options. This topic is discussed below. Other reserved words are used for properties status specification.

Example

Boolean: **true**, **false**
Numeric: **unlimited**

4. Types

string

A sting is a text put in apostrophes or quotation marks. If variables are applicable the string type parameter can be used in a variable.

regexp

A string where one can use regular expressions. If variables are applicable the regexp type parameter can be used in a variable.

uint

Unsigned integer or reserved word '**unlimited**' to define an infinitely big parameter.

bool

Boolean value. Has a value of either '**true**' or '**false**'.

identifier

An identifier is a text starting with a letter and having no spaces. Identifiers are used in the configuration file for specifying different objects: zones, connections, users, variables etc.

ip address

Numeric IP-address or Internet name put into apostrophes or quotation marks.

Example

```
123.48.132.56:12345
'canopus.aqua.comptek.ru:12345'
```

address range

Single IP-address or two addresses that form a range of addresses.

Example

Record	Description
19.38.12.45	Single IP-address.
19.38.12.45-19.38.12.49	Range.
19.38.12.45-49	Simplified range

direction

Reserved word '**incoming**' or '**outgoing**' or empty string (if allowed) for both directions.

side

Reserved word '**caller**' or '**callee**' or empty string (if allowed) for both sides.

bandwidth

Bandwidth in bits per second. Should be divisible by 100. A usage of 'K', 'M', 'G' modifiers makes it possible to specify the bandwidth in kilobytes, megabytes and gigabytes correspondingly. For example: 128K, 6.4K, 10M, 0.5M.

time

Date and time in the following format: Mon 31/12/2000 23:59:59. When using group expressions any of the values used in the expression can be substituted with '**.**' character. Some of the fields can be skipped to shorten the record:

- Mon 1/12/1999 12:00:00
- Mon 1/12/1999 12:00
- 1/12/1999 12:00:00
- 1/12/1999 12:00
- 12:00:00
- 12:00
- Mon

period

Two date & time records separated by '**-**' character form a time period.

Example

Expression	Description
Mon-Wed	From Monday to Wednesday weekly
12:00-17:59	From 12:00:00 to 17:59:59 daily.
1/./. 00:00-10/./. 23:59	From the 1st day 00:00:00 till 10th day 23:59:59 monthly
./1/. 00:00-./2/. 23:59	From 1st of January till 29st of February yearly

5. Basic options

system { ... }

Complex expression that contains basic **system options**.

monitor { ... }

Monitor options control expression. See **monitor options** chapter.

log { ... }

Complex expression containing **log control options**.

aaa { ... }

Contains RADIUS-server information and **AAA-options**.

zone <*identifier*> { ... }

Information zone declaration. Usually, it is a network segment that carries a data stream and is bandwidth-independent from other zones. This option is used for the telephone traffic limitation going through the specific network segment and other for specifying QoS parameters. The gatekeeper restricts all connections in the zone with no bandwidth available. See "**zone and connections options**" chapter with expressions description which can be included in this complex expression.

link [*identifier*] between <*identifier zone*> and <*identifier zone*> { ... }

This expression has the same functionality as '**zone**' expression with an additional feature to connect two zones for real network topology imitation. See "**Zone and connections options**" chapter with expressions description which can be included in this complex expression.

group <*identifier*> { ... }

A complex expression used to declare the group and its behavior. This expression can contain any expressions from '**user**' and '**gatekeeper**'. See "**user options**" and "**gatekeeper options**" chapters. You can create a group with a reserved word '**everyone**', the users declared below become members of this group.

user <*identifier*> { ... }

The meaning of this expression is close to the '**group**' expression. The only exception is that it declares a single user which can be registered in the system. See "**user options**" chapter with expressions description which can be included in this complex expression.

gatekeeper <*identifier*> { ... }

This expression is used for the user with extended possibilities declaration (gatekeeper type). It can contain both simple expressions of user and group and additional expressions. The main difference between '**user**' and '**gatekeeper**' expressions is that gatekeeper is not making a call right after the number match occurs.

The call is preceded by the Location Request (LRQ) message to the remote gatekeeper (or gatekeepers if more than one gatekeeper matched the route metric) in order to allocate the callee subscriber. Being a subscriber the remote gatekeeper should register in order to post its active state. If a remote gatekeeper does not support the registration, one should use '**static**' expression. For the detailed information see "**gatekeeper options**" chapter.

6. System options (system complex expression)

identifier <*string*>

Gatekeeper identifier. Used as a major identifying key of a gatekeeper. This identifier can be used by the subscribers when searching the gatekeeper or performing a registration on it when the subscriber intends to register on the gatekeeper with a specific name. Here, all requests for the search and

registration with incorrect gatekeeper identifier are rejected by the gatekeeper. However, requests with no gatekeeper identifier are processed.

address <*ip address*>

Output interface through which the gatekeeper is working. It is of a great importance to configure this parameter properly on the computers that have more than one interface, so that gatekeeper could use a right access to the H.323 network. This rule is not relevant when the computer has only one interface. However, if a host has the gatekeeper with several interfaces (e.g. acts as a router) and if this expression is skipped, the gatekeeper randomly chooses one of the interfaces' addresses and this might cause problems.

RAS port <*uint*>
default value: 1719

Port number used for RAS communication with a gatekeeper. If this expression is skipped, a standard 1719 port is used.

call signal port <*uint*>
default value: 1720

A port number which is used for a signaling level. Signaling port is used by other devices incoming requests which do not support a feature of the gatekeeper registration. If this expression is skipped a standard 1720 port is used.

7. Monitor options (monitor complex expression)

port <*uint*>
default value: 2040

Port number used by the Monitor

password <*string*>

Access password for the Monitor.

Note: this password is used for the configuration encryption when transmitting over the network using monitor protocol. If the password is not specified, no encryption is performed.

address <*address range*> [*bool*]
address <*bool*>

Defines a right to connect to a monitor service from a specific address or a set of addresses. Skipped address implies all possible addresses. Skipped "bool" parameter implies '**true**' value.

8. Log options (log complex expression)

file <*string*>

A path to a log file which contains a dump of the console output. Identical to the command prompt key '**consolefile**'.

[*direction*] ras messages <*bool*>

Controls RAS-messages output. According to the direction used incoming or outgoing messages are outputted to the log. The option controls all RAS-messages if the direction specification is skipped.

[*direction*] q931 messages <*bool*>

Controls Q.931-messages output.

[*direction*] h245 messages <*bool*>

Controls H.245-messages output.

[*direction*] radius messages <*bool*>

Controls RADIUS-messages output.

messages dump <*bool*>

Dump output for all incoming messages.

registrations <*bool*>

Registrations output.

connection status <*bool*>

Connection progress information output.

chosen route <*bool*>

Selected route output on alias resolve.

available bandwidth <*bool*>

Call available bandwidth output.

bandwidth changes <*bool*>

Used bandwidth changes output.

9. AAA option (AAA complex expression)

file <*string*>

File path that contains connections information, start time and their duration (CDR file). If the path is skipped, it is created in the folder with the configuration file. CDR file is not created if the option is skipped.

address <*ip address*>

RADIUS server address.

authentication port <*uint*>
default value: 1812

Authentication port. If this option is skipped a standard 1812 port is used.

accounting port <*uint*>
default value: 1813

Accounting port. If this option is skipped a standard 1813 port is used.

password <*string*>

RADIUS server password.

log failed calls <*bool*>
default value: false

Forces unsuccessful calls accounting in CDR-file and on the RADIUS server.

time zone name <*string*>
default value: "UTC"

Defines time zone name used in RADIUS packets.

time zone offset <*offset*>
default value: GMT

Time zone offset for AAA records. Allowed values: **GMT**, **local**.

10. Zone and connection options (zone & link complex expressions)

bandwidth <*bandwidth*>
default value: unlimited

Total zone bandwidth. Gatekeeper restricts connection establishing if the bandwidth exceeded.

connections <*uint*>
default value: unlimited

Number of connections that can be processed in the zone. Gatekeeper restricts a new connection establishing after this threshold is reached.

11. User options (user complex expression)

proxy level <*level*>
default value: signalling

Specifies a proxy level. Available values: '**none**', '**signalling**' and '**full**'.

- **none** – a subscriber can make and receive calls from other subscribers directly (using "direct" call type). However, in this mode the subscriber can request for the "routed" call type (with '**proxy level choice**' option enabled) which corresponds with '**signalling**' proxy level. This option is not recommended for usage as the call existence can be defined using IRR messages. That is why call duration information may not be precise.
- **signalling** – gatekeeper will use proxy for both "call signal" and "control" signaling levels. All calls of "direct" type are rerouted on the gatekeeper and the call type is changed to "routed".
- **full** – in addition to '**signalling**' proxy level, a gatekeeper will proxy all RTP channels with useful data.

If the subscribers that take part in the connection have different proxy levels, the highest is accepted. Using this feature you can set up '**signalling**' proxy level for the subscribers in the network whereas outer gateways use '**full**' level in order to avoid firewall.

proxy level choice <bool> default value: false

Controls the subscriber's possibility to change the proxy level. H.323 standard sets subscriber's possibility to set "direct" or "gatekeeper routed" mode. Thus, with '**proxy level choice**' option enabled the subscriber can switch from '**signalling**' or '**full**' proxy levels to '**none**' or stay in the current mode. If the subscriber is configured for proxy level '**none**', it can switch to '**signalling**' mode or stay in '**none**' mode.

hunt level <*level*> default value: neutral

Sets a hunting level if the subscriber is a caller.

Available values: '**none**', '**neutral**', '**soft**', '**hard**'.

- **none** – no hunting is performed in this mode. If for some specific reason the call cannot be made (remote subscriber is busy or remote equipment failed etc), the caller is not switched to another alternative and he will hear the error signal.
- **neutral** – the caller is switched to the alternative route in case of errors during the connection. However, if by this moment the data channels are established the switch is not performed. The gatekeeper is not attempting to postpone the data channels establishing – that is why this mode is called "neutral".
- **soft** – In this mode the gatekeeper attempts to postpone data channels establishing by moving FastStart and H245Address information into Alerting message.
- **hard** – In this mode the gatekeeper attempts to postpone data channels establishing by moving FastStart and H245Address information into Connect message.

Note that in any mode the subscribers have a possibility to open data channels using H.245 tunneling thus making the hunting impossible.

fast start <bool> default value: true

Controls "fast connect" procedure for the subscriber. Some subscriber units (e.g. Cisco ATA186) can be put to a non-working "fast start" state sequence; you can avoid it by restricting "fast connect" procedure for these subscribers.

h245 tunneling <bool> default value: true

Controls H.245 tunneling for the subscriber.

cost <uint> [at] default value: unlimited

Route cost. If there is more than one subscriber with the same alias, the subscriber with less cost is chosen. This expression can contain sub-expression '**at**'.

connections <uint> [at] default value: unlimited

Declares maximum number of connections that are simultaneously open to the user. This value is used in the subscriber's load calculations in the load balancing

procedure. If there is more than one subscriber with the same alias and price, the subscriber with less load is chosen. Anyway, the gatekeeper restricts a larger number of connections than declared in the command.

location <*identifier* ***zone***>

Subscriber's zone declaration. Is used for QoS and traffic shaping. If the expression is not defined, the subscriber is out of any zone.

bandwidth <*bandwidth*>
default value: 128k

Bidirectional and divisible by 100 bandwidth for static subscribers. For example, if the subscriber uses 64 kbit aLaw codec for both directions, the bandwidth should be specified as 128k.

login <*regexp*> [*bool*] [*set list*]
login <*bool*> [*set list*]

The alias which is used for subscriber's registration. Skipped "regexp" parameter implies all possible aliases. Skipped "bool" parameter implies '**true**' value. Using '**set**' sub-expression one can extract various parts of the alias.

Example

```
login 'user(.+)goinin' set UserAlias to '\1';
alias UserAlias;
```

The user that registers with 'userJohngoinin' will be registered with default user parameters and will be available via 'John' alias.

alias <*regexp*> [*bool*] [cost <*uint*>] [translate to <*regexp*>] [*set list*] [*af*]
alias <*bool*> [cost <*uint*>] [*set list*] [*af*]

Defines a set of actual aliases that are used when calling the subscriber. "bool" parameter defines the user's ownership for this alias. If "bool" parameter is set as '**false**', this alias is excluded from the list of actual aliases.

Example

```
alias '81234' false;
alias '8\d{5}' true;
```

In this example the subscriber can be reached via all 5-digit numbers starting with '8' excluding 81234. Skipped "bool" parameter implies '**true**' value. Skipped "regexp string" parameter implies all possible aliases.

'**translate to**' sub-expression is used for received dialed string translation when submitting it to the subscriber, like it is done in '**translate alias**' directive.

'**cost**' sub-expression sets a cost of the route to the individual alias. If this sub-expression is skipped, the route cost for the subscriber is used.

translate [*direction*] [*side*] alias <*regexp*> to <*regexp*> [*af*]

Alias translation for further processing and submission. "Direction" and "side" parameters control "incoming/outgoing" aliases and "caller/callee" aliases translation correspondingly. Incoming alias is translated as soon as it matches the declared alias for the subscriber. Alias is modified only with first matched translation. "Direction" and "side" parameters can be skipped in order to perform all aliases translation in the skipped class. For example, if caller and callee aliases are in the same numeration plan, you need to write the following translation rules in order to coordinate the work with the outer gateway:


```
translate incoming alias '(\d{3})' to '+73432451\1';
translate outgoing alias '+73432451(\d{3})' to '\1';
```

It is also very useful to assign an outgoing alias to the specific subscriber:

```
translate outgoing caller alias '.*' to '212';
```

whereas his actual alias in configured as

```
alias '212';
```

dial [*identifier*] <*bool*> [*at*]

Configures the user's right to call another user or the user from the group. Skipped identifier implies all users in the system. By default the user can call any subscriber in the system. If the user or its group has one or more '**dial**' expressions, the calls to the users that are not declared by these expressions are refused. Instead of a username one can use '**self**' reserved word.

address <*address range*> [*bool*]
address <*bool*>

Configures the right of the user to register in the system from the specific address or a range of addresses. Skipped range of addresses implies all possible addresses. Skipped boolean value implies '**true**' value. However, at least one the parameters must be present. By default the user has a right to register from any address and '**address true**' expression is equivalent for this setting. If the user or its group contain one or more '**address**' expressions, all attempts to register from any other addresses are be rejected. If the user is static (the user is marked with '**static**' expression), '**address**' expressions define a set of allowed addresses for the direct gatekeeper signal call. '**Static**' expression also implies '**address**' expression with the address and access flag set to '**true**' automatically.

static [*ip address*]
static [*bool*]

This expression lets the subscriber to register automatically with given a call signal address. It is used for the terminals that cannot communicate with a gatekeeper. All RAS registrations for this account are rejected. This expression also implies '**address**' expression with the address and access flag set to '**true**' automatically. If "IP address" parameter is skipped the calls on/from this subscriber are not allowed. However, in order to let the user to make calls you can specify '**address**' expression.

Example

```
address 19.38.12.7;
static;
```

In this example the static subscriber can make calls but cannot receive them.

Example

```
address 19.38.12.7 false;
static 19.38.12.7;
```

In this example the subscriber can receive calls but cannot make them.

registration validity <*uint seconds*>
default value: unlimited

This expression limits subscriber's registration validity up to "uint" seconds. The subscriber should support re-registration ability in order to stay active. If the subscriber does not re-register in the specified period of time, it will be

disconnected by the system. Use '**unlimited**' reserved word in order to cancel the specified time period (e.g. in the user group).

connection validity <*uint seconds*> default value: 60

This expression limits connection durability up to "uint" seconds. The subscriber should keep the active state by periodical sending of IRR message. IRR period is set by the gatekeeper and is few times less than **connection validity**' parameter. Gatekeeper does not request IRR messages from the subscriber if '**connection validity**' is set as '**unlimited**'. This feature helps to connect those subscribers which do not send IRR messages during the call despite of the fact that this function is declared to be obligatory in H.323 standard. It is NOT recommended to use '**connection validity unlimited;**' with '**proxy level none;**' because in this case there is no reliable way for the gatekeeper to learn that the connection was interrupted. This causes imprecise data for billing and "buzzed" connections.

max [*direction*] ringback duration <*uint seconds*> default value: unlimited

Answer awaiting duration in seconds. The least value among incoming for the terminating side and outgoing for the caller side is taken.

max [*direction*] connection duration <*uint seconds*> default value: unlimited

Maximum connection duration in seconds. The least value among incoming for the terminating side and outgoing for the caller side is taken.

radius authentication <*mode*> default value: none

Enables RADIUS-server authentication mode. Available values: '**none**', '**login**', '**caller**', '**callee**', '**call**' and '**full**'. These options can be used together with '|' character.

- **none** – no authentication is performed.
- **login** – the gatekeeper requests the RADIUS-server for user's authentication
- **caller** – the gatekeeper requests the permission for the connection establishing if the subscriber is a caller
- **callee** - the gatekeeper requests the permission for the connection establishing if the subscriber is a callee.
- **call** – identical to 'caller | callee'.
- **full** – identical to '**login** | **call**'.

The identifying request for authentication has a "Service-Type" set to "Login". The identifying request for connection establishing has a "Service-Type" set to "Call Check".

radius accounting <*mode*> default value: none

Enables accounting mode on the RADIUS-server. Acceptable values: '**none**', '**login**', '**caller**', '**callee**', '**call**' and '**full**'. These options can be put together using '|' sign.

- **none** – no accounting is performed for the subscriber.
- **login** – gatekeeper performs user's presence time accounting in the system.

- **caller** – the gatekeeper performs calls duration accounting if the subscriber is a caller.
- **callee** – the gatekeeper performs calls duration accounting if the subscriber is a callee.
- call – identical to 'caller | callee'.
- **full** – identical to '**login** | **call**'.

The request for authentication accounting has a "Service-Type" field set to "Login". The request for connection accounting does not have "Service-Type" field.

radius name <*string*>
default value: UserName

Specifies a username for RADIUS operations. Also one can use build-in variables UserName or H235Name which contain username from the configuration and username from supplied H.235 information correspondingly.

radius password <*string*>

Specifies a password for RADIUS operations. Zero length password implies no password usage – it is a default setting. If '**radius password** H235Password' expression is used that means that the gatekeeper extracts the password from the H.235 information supplied by the user. This mode is supported only by Cisco Access Token.

h235 authentication <*bool*>

Enable/disable H.235 RAS and Q.931 messages authorization. Supported methods: CAT, Password with hashing (MD5).

h235 name <*string*>
default value: UserName

Specifies a name which should be supplied by the user for messages authorization.

h235 password <*string*>

Specifies a password which should be supplied by the user for messages authorization.

display <*string*>
default value: UserDisplay

Assigns a specified value to the Q.931 display field. Original '**display**' field value is put to the UserDisplay variable automatically. You always can restore the original mode by executing '**display** UserDisplay' command.

connection dupe <*bool*>
default value: false

Allow/restrict the user to create connections with duplicate connection identifier. Normally, this function is restricted. However, some subscribers, like rerouting answering machines, can open a connection with changed address but with the same connection identifier. Usually, this call is restricted by the loop detector; thus, in this case, you should enable '**connection dupe**' function for this subscriber. Be careful when using this option; the subscriber should not act twice from one side (e.g. caller side) in a different connection with the same

connection identifier. It is caused by the fact that the gatekeeper in this case is not able to differentiate these connections.

set <identifier:variable> to <string>

Assigns a string value to the variable. Only the first '**set**' expression for the specific variable has the power.

make login <regexp> <set list>

Makes a variable according to an alias. Note that this expression does not influence on the registration process; it is turned on only after the alias came through '**login**' expression.

Example

```
group everyone
      make login '.*' set UserAlias to '\0',
      alias UserAlias;

user User1 login '201';
user User2 login '202';
```

In this example User1 is registered only with "201" alias. The actual alias is "201".

make [direction] [side] alias <regexp> <set list> [af]

Enables the system to make a variable corresponding to the call alias. Skipped direction and/or side parameter imply all aliases of all types in the skipped class.

alternate gatekeeper <ip address> [string] [gatekeeper ...]
alternate gatekeeper none

Declares a list of alternative gatekeepers for the user. This list is sent to the subscriber if he supports alternative gatekeeper procedures by including a SupportsAltGK field into RRQ message. '**alternate gatekeeper none**' expression disables this function – it is a default value.

12. Gatekeeper options (gatekeeper complex expression)

ras address <ip address>

Declares a RAS level address of a remote gatekeeper. It is used with '**static**' option if your gatekeeper cannot locate this address from other sources.

identifier <string>

Remote gatekeeper identifier. The gatekeeper uses this identifier in all requests where it is possible. If '**ras address**' expression is skipped, the gatekeeper tries to locate the remote gatekeeper using Gatekeeper Request (GRQ) message. Note: GRQ is a broadcasting message and does not go through routers.

register alias <string> [alias ...]
register alias none

Forces the gatekeeper to register on a remote gatekeeper in order to show its active status. Registration method is the same as for the subscribers. The remote gatekeeper can interpret our gatekeeper as a subscriber if it does not analyze the registering terminal type.

link h235 name <*string*> default value: UserName

Sets a H.235 name for the remote gatekeeper connection.

link h235 password <*string*>

Sets a H.235 password for the remote gatekeeper connection.

link h235 authentication <*bool*> default value: false

Enables/disables H.235 authentication when connection to the remote gatekeeper.

registration period <*uint seconds*>

Forces the gatekeeper to repeat registration procedure in the specified time period ("uint" seconds). This function is used for the active state acknowledgment.

link alternate gatekeeper <*bool*> default value: false

Allows alternative gatekeeper procedures during current connection with the remote gatekeeper. When enabled, SupportsAltGK flag is raised, alternative gatekeepers list provided by the remote gatekeeper is analyzed and all procedures for switching between alternative gatekeepers are enabled.

13. Sub-expressions.

at <*period*>

Sets the expression validity period.

set <*identifier variable*> to <*string*>

Assigns a string value to the variable.

Example

```
set UserLogin to 'user(.+)';
login UserLogin set UserAlias to '\1';
alias UserAlias;
```

set <*identifier variable*> add <*string*>

Adds a string to the specified variable. If the variable already contains non-zero length string, the new string will be preceded by the '|' sign. This function is specially implemented for several aliases processing when registering the subscriber. If the string which carries the variable contains brackets, the substring is inserted before the first closing bracket.

Example

```
set UserAlias to 'common|(2())';
login 'gw5(\d{3})' set UserAlias add '\1';
alias UserAlias;
```

In this example, the user with "gw5100" and "gw5101" aliases, has a resulting alias of "common|(2(100|101))".

set list

Sub-expression lists '**set** ... **to**' and '**set** ... **add**'.

Example

```
alias '(\d{3})(\d{3}).*' set Account to '\1'
                         set PIN to '\2';
```

VI. Supplementary information

1. CDR file format

CRD-file name is specified in **aaa** section using **file** expression and has the following format.

Each string corresponds with a separate call, information fields have "," separator. Below are the fields in the order of their presence in the CDR file.

№	Description	Example
1	Log record data. Always in GMT+0	03/06/2004 12:13:33
2	Caller subscriber identifier	abonent1
3	Call Signalling address from which the call was initiated	192.168.0.3:3175
4	Callee subscriber identifier	abonent2
5	Call Signalling address where that call is terminated	192.168.0.4:1720
6	Conversation start date, depends on **aaa time zone offset** setting	03/06/2004 17:58:34
7	Conversation end date, depends on **aaa time zone offset** setting	03/06/2004 18:13:33
8	Conversation duration	0/0:14:59.365
9	Call completion code, ITU-T Q.850	16
10	Caller subscriber aliases, separated by «&» sign, if there are several aliases	1001 & abonent1
11	Callee subscriber aliases, separated by «&» sign, if there are several aliases	73431111111

2. SGCMonitor protocol

TCP connection is open on the 2040 port and all packets are send with TPKT envelop. Using **log monitor messages yes** expression, one can observe packets sent in the system log.

```
--
-- Copyright (c) 1999-2003 Aqua Project Group
-- ASN.1 Gatekeeper monitor protocol version 1.00
--


MonitorProto DEFINITIONS AUTOMATIC TAGS ::=
BEGIN


-------------------------------------------------------------
-- Base types
-------------------------------------------------------------

IpAddress ::= SEQUENCE
{
  ip        OCTET STRING( SIZE( 4 ) ),
  port            INTEGER( 0..65535 )
}

Date ::= SEQUENCE
{
  year            INTEGER,
  month           INTEGER( 1..12 ),
  day       INTEGER( 1..31 )
}

Time ::= SEQUENCE
{
```

```
  hour            INTEGER( 0..23 ),
  minute    INTEGER( 0..59 ),
  second    INTEGER( 0..59 )
}

DateTime ::= SEQUENCE
{
  date            Date,
  time            Time
}

MessageIdentifier    ::= INTEGER( 0..65535 )
GatekeeperIdentifier ::= BMPString( SIZE( 1..128 ) )
EndpointIdentifier   ::= BMPString( SIZE( 1..128 ) )
ZoneIdentifier       ::= OCTET STRING( SIZE( 1..128 ) )
ConferenceIdentifier ::= OCTET STRING( SIZE( 16 ) )
Bandwidth  ::= INTEGER( 0..4294967295 ) -- in 100s of its
CallReferenceValue   ::= INTEGER( 0..65535 )

SegmentType ::= ENUMERATED
{
  comprehensive,
  initial,
  intermediate,
  final
}

CompletionCode ::= SEQUENCE
{
  errorText             OCTET STRING OPTIONAL
}

EndpointType ::= CHOICE
{
  gatekeeper            NULL,
  gateway         NULL,
  mcu             NULL,
  terminal        NULL
}

ProxyLevel ::= CHOICE
{
  none                  NULL,
  signalling            NULL,
  full                  NULL
}

ChallengeValue ::= OCTET STRING( SIZE( 8 ) )
ChallengeResponse ::= OCTET STRING( SIZE( 16 ) )

Challenge ::= SEQUENCE
{
  value                 ChallengeValue OPTIONAL
}

RequestDataClass ::= ENUMERATED
{
  gatekeeper,
  zone,
  endpoint,
  connection,
  configuration,
  log,
  ...
```

```
}

RequestData ::= SEQUENCE
{
  dataClass       RequestDataClass,
  fetchCurrentData     BOOLEAN,
  notifyChanges         BOOLEAN,
  ...
}

RequestResultError ::= SEQUENCE
{
  code                 INTEGER( 0..127,... ),
  text                 OCTET STRING OPTIONAL
}

RequestResult ::= CHOICE
{
  ok              NULL,
  okDataFollowOn  NULL,
  error                RequestResultError
}

-------------------------------------------------------
-- Gatekeeper
-------------------------------------------------------

GatekeeperInfo ::= SEQUENCE
{
  identifier            GatekeeperIdentifier,
  version         OCTET STRING,
  osInfo          OCTET STRING,
  rasAddress            IpAddress OPTIONAL,
  csAddress       IpAddress OPTIONAL,
  ...
}

-------------------------------------------------------
-- Zone
-------------------------------------------------------

Zone ::= SEQUENCE
{
  identifier            ZoneIdentifier,
  bandwidth       Bandwidth,
  occupiedBandwidth     Bandwidth,
  connections           INTEGER,
  occupiedConnections   INTEGER,
  links                 SEQUENCE OF ZoneIdentifier,
  ...
}

-------------------------------------------------------
-- Endpoints
-------------------------------------------------------

Endpoint ::= SEQUENCE
{
  identifier            EndpointIdentifier,
  userName        OCTET STRING,
  loginAliases SEQUENCE OF BMPString( SIZE( 1..256  ),
  aliases         SEQUENCE OF BMPString( SIZE( 1..256 ) ),
  rasAddress            IpAddress OPTIONAL,
  csAddress       IpAddress,
```

```
  endpointType          EndpointType,
  zone                   ZoneIdentifier OPTIONAL,
  static          BOOLEAN,
  ...
}

-------------------------------------------------------------
-- Connections
-------------------------------------------------------------

Connection ::= SEQUENCE
{
  identifier             ConferenceIdentifier,
  callerIdentifier     EndpointIdentifier OPTIONAL,
  callerCSAddress IpAddress OPTIONAL,
  calleeIdentifier     EndpointIdentifier OPTIONAL,
  calleeCSAddress IpAddress OPTIONAL,
  occupiedBandwidth    Bandwidth,
  conversationStart    DateTime OPTIONAL,
  conversationStop     DateTime OPTIONAL,
  zones                SEQUENCE OF ZoneIdentifier OPTIONAL,
  callerAliases        SEQUENCE OF OCTET STRING OPTIONAL,
  calleeAliases        SEQUENCE OF OCTET STRING OPTIONAL,
  callerDisplay        OCTET STRING OPTIONAL,
  calleeDisplay        OCTET STRING OPTIONAL,
  proxyLevel           ProxyLevel,
  ...
}

-------------------------------------------------------------
-- Main
-------------------------------------------------------------

ServerRequest ::= SEQUENCE
{
  identifier           MessageIdentifier,
  body                 CHOICE
  {
    challenge          Challenge,
    ...
  }
}

ServerResponse ::= SEQUENCE
{
  identifier           MessageIdentifier,
  segmentType          SegmentType DEFAULT comprehensive,
  body                 CHOICE
  {
    requestResult RequestResult,
    gatekeeperInfo     GatekeeperInfo,
    configurationData  OCTET STRING,
    logData       OCTET STRING,
    zones         SEQUENCE OF Zone,
    endpoints           SEQUENCE OF Endpoint,
    connections         SEQUENCE OF Connection,
    ...
  }
}

ServerNotification ::= CHOICE
{
  gatekeeperInfo  GatekeeperInfo,
  logData         OCTET STRING,
```

```
  zonesAdded            SEQUENCE OF Zone,
  zonesChanged          SEQUENCE OF Zone,
  zonesRemoved          SEQUENCE OF ZoneIdentifier,
  endpointsAdded  SEQUENCE OF Endpoint,
  endpointsChanged      SEQUENCE OF Endpoint,
  endpointsRemoved      SEQUENCE OF EndpointIdentifier,
  connectionsAdded      SEQUENCE OF Connection,
  connectionsChanged    SEQUENCE OF Connection,
  connectionsRemoved    SEQUENCE OF ConferenceIdentifier
}

MonitorServerMessage ::= CHOICE
{
  request         ServerRequest,
  response        ServerResponse,
  notification          ServerNotification
}

ClientRequest ::= SEQUENCE
{
  identifier            MessageIdentifier,
  segmentType           SegmentType DEFAULT comprehensive,
  body                  CHOICE
  {
    requestData         RequestData,
    setConfigData OCTET STRING,
    restart       BMPString( SIZE( 1..25 ) ), -- Place "Yes,
please!" here.
    reloadConfiguration BMPString( SIZE( 1..25 ) ), -- Place
"Yes, please!" here.
    removeEndpoints     SEQUENCE OF EndpointIdentifier,
    removeConnections   SEQUENCE OF ConferenceIdentifier,
    ...
  }
}

ClientResponse ::= SEQUENCE
{
  identifier            MessageIdentifier,
  body                  CHOICE
  {
    challengeResponse   ChallengeResponse,
    ...
  }
}

MonitorClientMessage ::= CHOICE
{
  request         ClientRequest,
  response        ClientResponse,
  ...
}

END -- of ASN
```

3. Basic POSIX regular expressions

Regular expressions language is a conventional record for text patterns creating. Usually regular expressions take part in the comparing procedure with strings in order to find a match between the sting and the pattern or for the substring search.

This chapter describes POSIX record for the regular expressions. It is not an official review but written in a simple and a clear way.

Getting started with regular expressions

In its most simple form, the regular expression is a string containing characters which should match in a single case only. For example:

```
regexp
```

matches "regexp" string only.

Some characters in the regular expressions have a special meaning. They are not compared character by character as it was in the previous example, but describe a wider template. For example, "*" symbol is used to specify that the previous element of the regular expression can be repeated 0, 1 and more times. For example:

```
smooo*th
```

"*" points that the previous "o" character can be repeated 0 or more time. That means that this template matches with:

```
smooth
smoooth
smooooth
smoooooth
...
```

Sometimes there is a need to write a template where the "*" special character should match as a simple symbol – that means you do not want this character to act as a special one. In order to do that, mark this "*" character with back slash sign:

```
smoo\*th
```

matches with the string:

```
smoo*th
```

and no others.

Next paragraphs describe some other special characters that are used in the regular expressions.

Symbolic regular expressions

Symbolic regular expression is a string that does not contain any special characters. It matches only with the same string and no other. For example:

```
literally
```

matches with

```
literally
```

and nothing else.

Usually, space bar characters, digits and letter are not used as special characters. Some punctuation marks are special characters, some of them are not (please refer to the short reference given in the end of this chapter)

Character sets

This paragraph describes "." and "[" special characters..

"." Matches with any symbol excluding NULL. For example:

```
p.ck
```

matches with

```
pick
pack
puck
```

```
pbck
pcck
p.ck
...
```

"[" opens character set. Character set is close with "." character in the fact that it does not match with any specific character but with any character from the provided set. "[" is different from "." so that using "[" you can declare some definite character set.

Character set can be in three different forms.

In the first form it looks as follows:

[<character set>] – each character from the set.

In the second form the character set is marked by a special negation character and looks as follows:

[^<character set>] – each character NOT from the set.

<character set> - character set listing. It can be presented as a string from separate characters:

```
[aeiou]
```

or a characters range:

```
[0-9]
```

These two formats can be mixed:

```
[A-za-z0-9_$]
```

Special characters (like "*") do not have a special meaning in the character sets. "-" character, as shown above, is a special character if it is not the first one in the set.

Four characters set:

```
[-+*/]
```

The third form of character sets uses predefined character classes:

\<character class> -- each character described in the class.

Character classes supported:

Character	Description
\w	Digits and letters set
\d	Digits only
\l	Lowercase letters
\s	whitespace characters
\u	Uppercase letters

The class character set can be inverted by changing the class character from lowercase to the uppercase.

The character sets can be used in any spot of the regular expression just as where usual characters are placed.

Sub-expressions

Sub-expression is a regular expression but into brackets: (). The sub-expression can be used in any part of the regular expression.

Sub-expressions can be used for grouping the regular expression constructions. For example, the "*" character is applied to the previous character. Like this:

```
smooo*th
```

matches with


```
smooth
smoooth
...
```

Using sub-expressions we can apply "*" to a longer string:

```
banan(an)*a
```

matches with

```
banana
bananana
banananana
...
```

Sub-expressions also play an important roles in substitutions and references.

Repeated sub-expressions

Repeat operator "*" is applied to the previous character, the set of characters, sub-expression and to the reference. "*" means that the previous element can match 0 or more times.

```
bana(na)*
```

matches with

```
bana
banana
bananana
banananana
...
```

"+" character is similar to "*", but the difference is that "+" means that the previous character should be met at least one time:

```
bana(na)*
```

matches with

```
bana
```

but does not match with

```
bana(na)+
```

Both of these regular expressions match with

```
banana
bananana
banananana
...
```

Thus, bana(na)+ is banana(na)* for short.

Optional sub-expressions

"?" character means that the previous character, the set of characters or the sub-expression is optional. It may match or it can be skipped:

```
CSNY?
```

matches both with

```
CSN
CSNY
```

Numeric sub-expressions

Interval expression {m,n}, where m and n are non-negative integers and n >= m, is applied to the previous character, the set of characters, the sub-

expression or to the reference. It means that the previous element should match at least m times but not more than n times.

For example:

```
c([ad]){1,4}r
```

matches with

```
car
cdr
caar
cdar
...
caaar
cdaar
...
cadddr
cddddr
```

Alternative sub-expressions

These sub-expressions have a following format:

```
regexp-1|regexp-2|regexp-3|...
```

The expression matches with any of regexp-n. For example:

```
Crosby, Stills, (and Nash|Nash, and Young)
```

matches with

```
Crosby, Stills, and Nash
Crosby, Stills, Nash, and Young
```

References, extractions and substitutions

The reference is used in the form of \n. N is a decimal non-zero number. In order for the reference to be active, the regular expression should have at least N sub-expressions put into brackets.

The reference is a symbol by symbol match with what is in the corresponding sub-expression. For example:

```
(.*)-\1
```

matches with

```
go-go
ha-ha
wakka-wakka
...
```

Sometimes sub-expressions are used for the substitutions using references numeration. For example:

```
translate alias '90244(\d{,5})' to '90248\1'
```

firstly matches with

```
9024412345
```

sub-expression 1 matches with "12345". The command substitutes the matched string with "90248\1". After the substitution we get the following:

```
9024812345
```

Short reference for regular expressions syntax

abcd – matches with a corresponding string character by character.

"." -- matches with any character except NULL.

[a-z_?], ^[a-z_?], \d and \D – sets of characters

(sub-expression) – grouping into the sub-expression

The following special characters and their sequences can applied to the character, the set of characters, sub-expression or to the reference:

Symbol	Description
*	Previous element is repeated 0 or more times
+	Previous element is repeated 1 or more times
?	Previous element is repeated 0 and 1 time
{m,n}	Previous element is repeated at least m but not more than n times.
regexp-1|regexp-2|..	Matches with any from regexp-n.

Special characters like "." or "*" or "?" or "+" can be interpreted as a simple characters when placing back slash sign before them ("\").