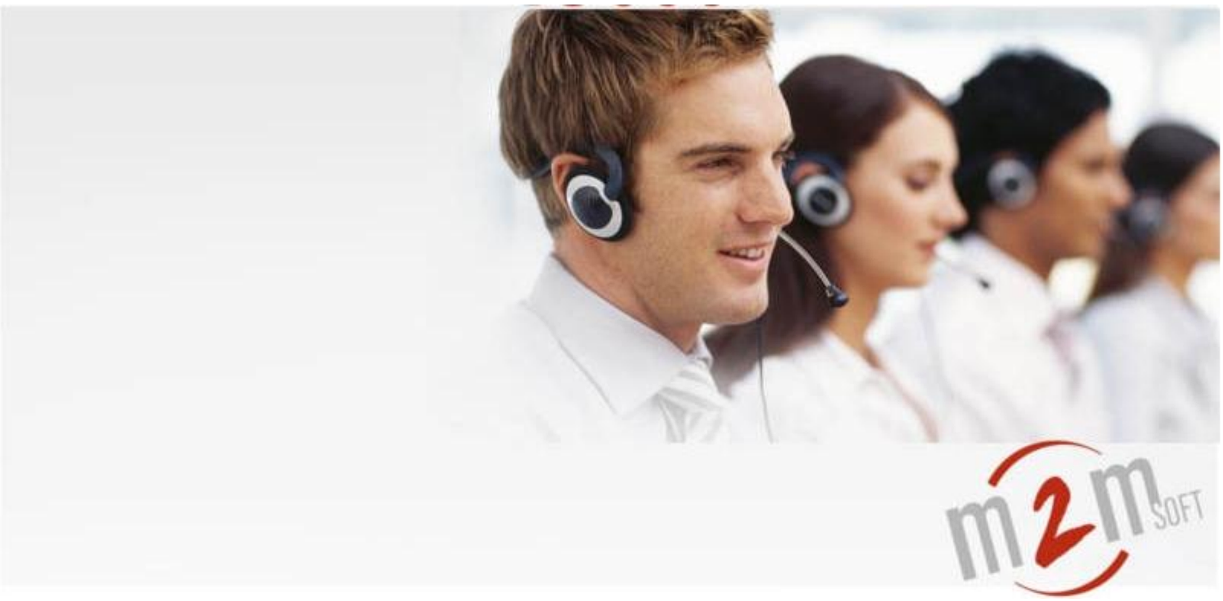
S5000

Voice and Video Softswitch & IPBX

Reference manual



Reference: s5000_US_RefManual
Version: 2.0-ed1
Date: June 17th 2013

The latest update of this manual is available at: http://www.m2msoft.com

M2MSOFT - 14 Rue de l'Europe, Parc d'Activités du Terlon, 31850 Montrabé. France
Phone: +33 820 200 263 – Fax: +33 561 50 02 32 – Email: contact@m2msoft.com - http://www.m2msoft.com

1. Content

2. Introduction

Thank you for your choice of the S5000 Softswitch and IPBX.
The S5000 allows you to set up and run IP voice and video communications within your network.
Operators can take advantage of the advanced routing capabilities.
Enterprise users can take advantage of the IPBX features.
Advanced users can easily develop complex new features with the S5000 APIs.

This manual contains all the necessary material:

- to install the product,
- to set up and run a working configuration with a number of phones and other equipment (Gateways, MCUs, other softswitches)
- to administer the system
- to develop new advanced services

Experienced users can take advantage of the development APIs to build and run advanced call control services and gateways, as described in the second part of this document.

This manual is not a VoIP, H323, SIP and other protocols guide and the reader is expected to have some basis on the subjects.

This manual applies for S5000 version 1.95r0 and above and API JGKXAPI version 1.16rc30 and above. Some features are available in selected version only. Contact M2MSOFT to upgrade your software if you need some new features.

Feel free to make us hear from you !

If any part of this manual appeals comments from you or if you do think some improvement can be made, we would be glad to hear from you to offer better manual quality and guidance.
Thanks in advance to report any suggestions to :

contact@m2msoft.com

or

Products support
M2MSOFT
14 Rue de l'Europe, Parc d'Activités du Terlon
31850 Montrabé
France
Call from France : 0820 200 263 (0,09 Euro TTC/min)
Call from other countries: +33 820 200 263

3. M2M-S5000 Concept

3.1. The GIMS framework

M2M-S5000 is part of a global framework solution called **GIMS** (Global IP Multimedia System) dedicated to provide users all necessary components to create and run voice, and video communication applications. GIMS contains the M2M-S5000 and includes other components such as programmable protocol stacks (H323 is one of them), C3000 audio/video Conference Bridge, A6000 audio servers… for example.

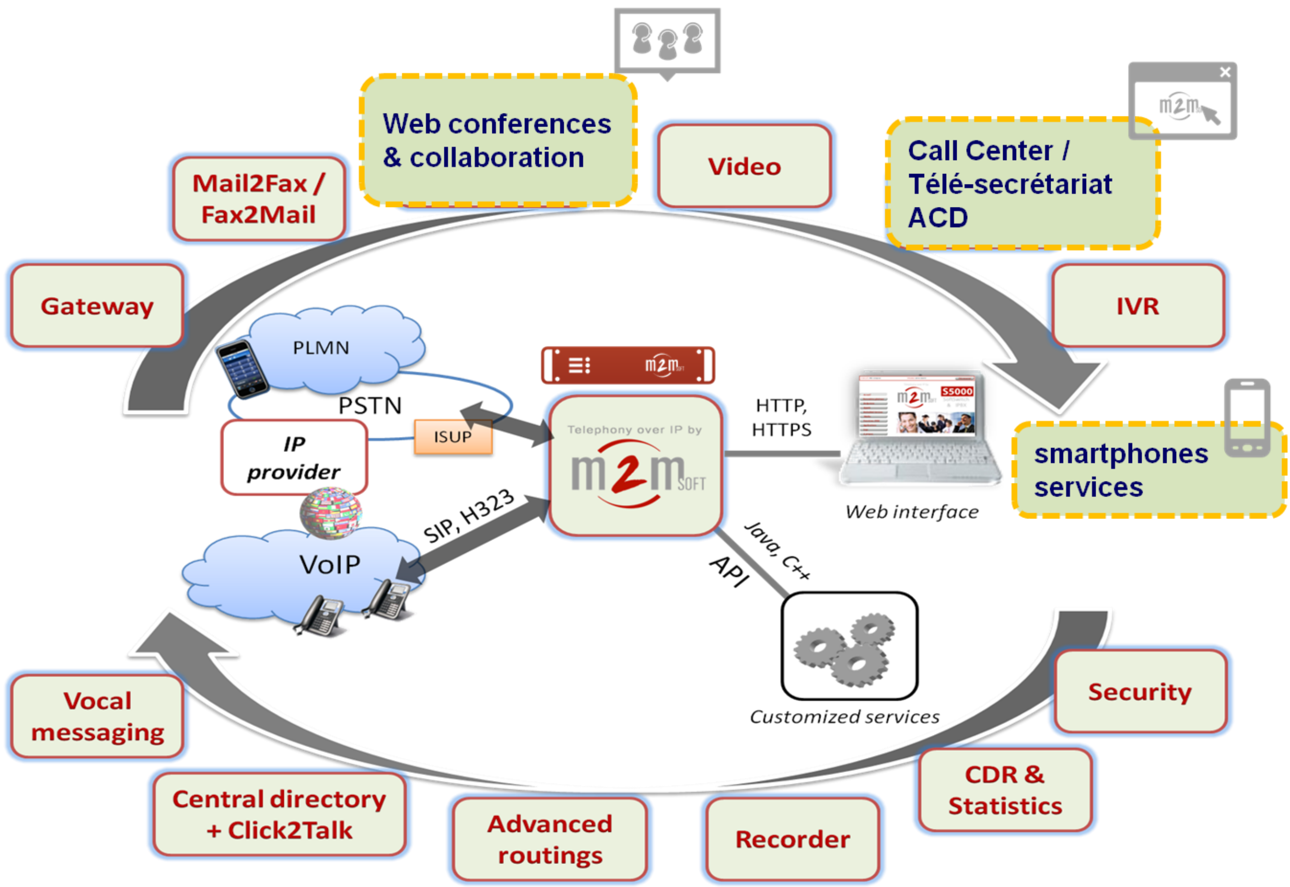
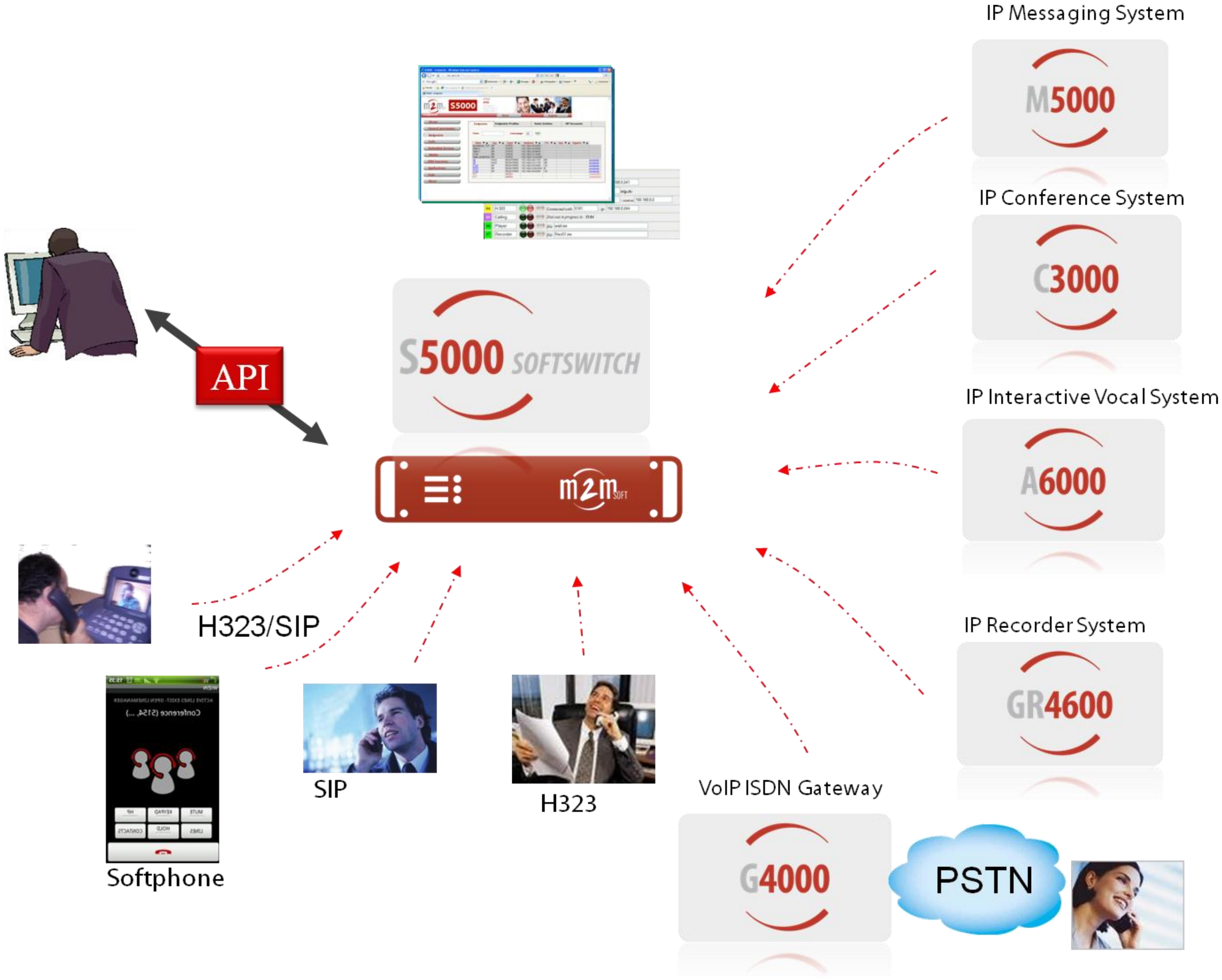


Fig.0. GIMS global offer and ability to serve a number of services



<u>Fig.1: a typical S5000 Softswitch centric system with a media server.</u>

A set of voice and video terminals are connected to the S5000 either through IP access or through an ISDN Gateway. Specialized components such as a conference bridge MCU **(C3000)**, an interactive vocal system **(A6000)**, a messaging system **(M5000)**, a recorder system **(GR4600)**, an ISDN/VoIP gateway **(G4000)** are used for special purpose on call termination.
This set is able to perform a rich range of services with and without the S5000 APIs and is called the GIMS architecture.
The following chapters describe the M2M-S5000.

3.2. S5000 features

M2M-S5000 is a solution for voice and video communications and services. A framework environment to **run** applications and handle voice/video equipment and a framework environment to **build** such applications.

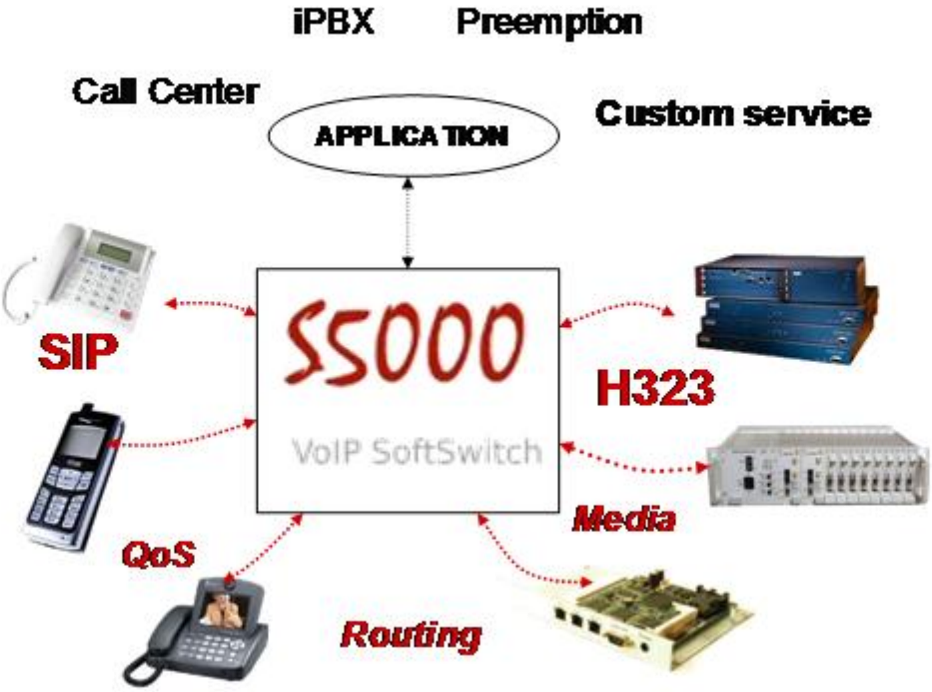


Fig.2 S5000 connects all a heterogynous environment

M2M-S5000 functional features are:

- Terminals/MCU/gateways H323 or SIP registration.
- Point to point VoIP calls between IP terminals, MCU, Gateways, mixed H323 and SIP calls.
- SIP registrar, proxy and B2BUA application server (product option).
- Signalling H225, H245, SIP-UDP/TCP/TLS flows routing and control.
- Media/RTP routing and Media Termination Points management (local and remote).
- T120 proxification.
- Embedded media server (for voice announcements).
- Advanced routing (load balanced trunks, backup, limitation, with busy and no answer control).
- Group redundancy and multicast discovery (for H323 and SIP).
- Forward selected calls or on no answer / busy / unavailable destination.
- Call transfer, call waiting.
- Called and calling numbers modifications.
- Multi ringing
- Unique feature of virtual lines IPBX system (1 physical line, N virtual lines on 1 media link)
- Call distribution thru queues
- Adjacent areas calls management through inter S5000 or inter proxies calls
- Secured calls through crypto signalling (and optionally media)
- Call detailed records generation per call
- Softswitch and IPPBX modes with Session Border Controller (SBC) functions (1-IP)
- JAVA and C Program Interface (API) to develop specific services that suit your needs.
- Embedded and secured Web administration interface.
- Multiplatform and Multi-OS.

Common supported hardware

Terminals

Microsoft Netmeeting 3.01
Microsoft Messenger 4.7 (SIP)
Siemens Optipoint 300 and 400, C470IP
Vcon MeetingPoint 4.51, Vpoint
OpenPhone (Equivalence Ltd)
XLite/Eyebeam/Bria
TipTel, YeaLink, Innovaphone 110/230
SwissVoice IP10 (H323, SIP)
Leadtek BVP8770/8750
Polycom ViewStation 512
SjLabs SJPhone (H323 & SIP)
Thomson ST2030 (SIP), ST2020, ST2022
GrandStream B2000 (SIP)
UtStarCom F1000 Wifi (SIP)
SNOM 320, 820
Panasonic
AAstra 53i, 55i, OMM/RFP32 DECT/SIP
Kirk Dect/SIP Phones

Gateways, MCUs and 3rd parties

GW Cisco AS5300, IOS 12.2 (8) T
GW Cisco 2600, IOS 12.2 (8) T2
GW Cisco ATA186
Cisco Call Manager Express
GW Motorola (or VANGUARDMS) – T2 v5.6
GW Quintum D3000
GW AudioCodes MP102/104/108/11X
GW AudioCodes Mediant1000/2000
Alcatel OmniPbx Enterprise
OpenMCU
Radvision OnLan, ViaIP
Polycom MGC100/50
ALCATEL OXE v6/v7/v8 and above in H323 and SIP
TAINET Venus 28xx
PATTON Smart Node 45XX

and others on request. Please see the release notes document for the latest updates.

M2M-S5000 technical features are:

- Support for protocols: H323v4, H225.0, RAS, Q931, H245, H450, T120, SIP (RFC3261, RFC2833, DTMF-INFO, and more) on UDP, TCP and TLS.
- RSVP (RFC2205, RFC2210, RFC2215) support for media channel reservations.
- TLS 1.0 support for secured calls and optional SRTP
- Routed modes H225.0/Q931 and H245 (can be disabled).
- H323 Fast-Start support.
- H245 Tunneling support.
- Support for Gatekeeper Discovery (GRQ).
- Support for H323Annex G, LRQ.
- Support for multicast SIP Registrar Discovery.
- Embedded services and per terminal services: forwards, call routing based on calling and called numbers, etc.
- Routing toward applications, load balanced trunks, alternate trunks…
- Virtual terminal support for embedded media calls (RTP connectors).
- Local and remote Media Termination Points management (RTP management).
- Session Border Controller Function with complete NAT and IP control and management (1 unique IP from the outside)
- SMS Short messages routing and generation(RFC3428)
- HTTP and HTTPS access to embedded web interface
- Multi-OS support: All flavors of Windows and LINUX (JAVA JVM 1.4 and above).
- Support for the main market endpoints.

3.3. S5000 layers

One can define three main functional layers within the S5000:

- a connectivity layer to handle the terminals and voice/video equipment connections
- a route & control layer with call routing definitions, terminals and calls authorizations definitions, bandwidth management, etc.
- a supervision layer with views on endpoints, calls, and applications
- the advanced services layer which enable to develop and run user applications on top of the S5000 with GKXAPI.

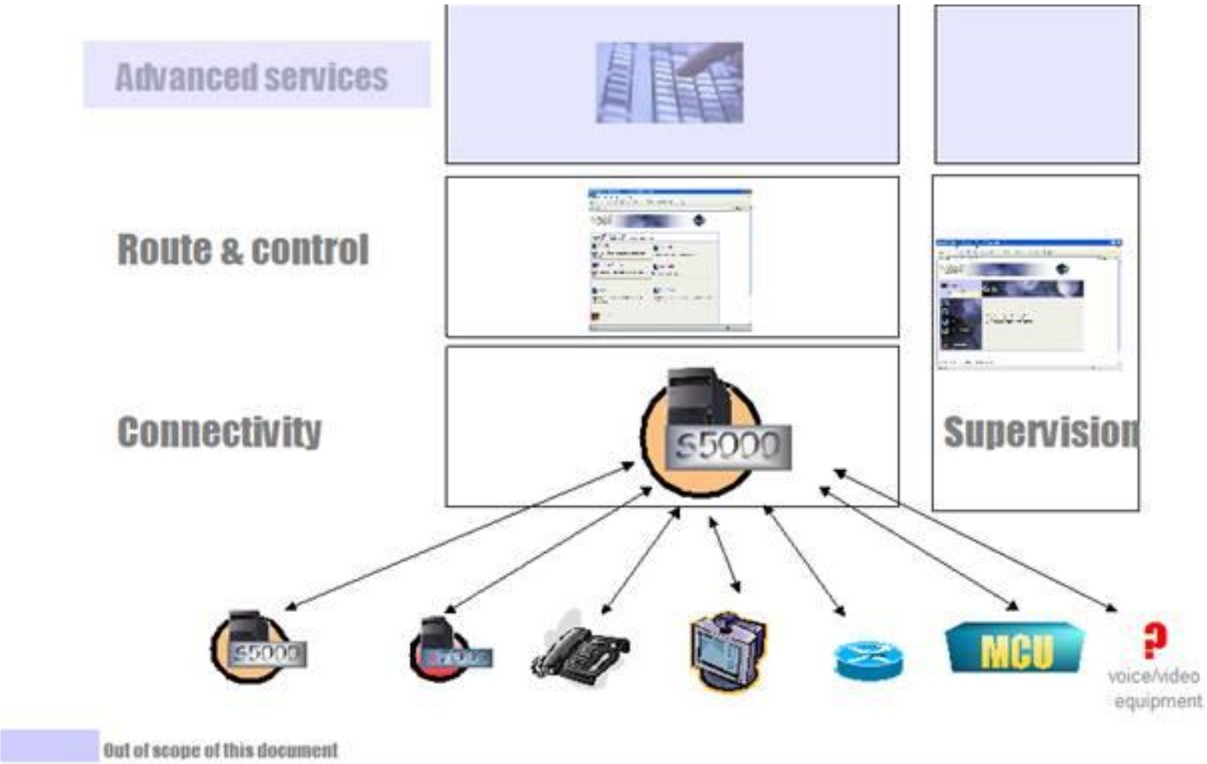


Fig.3 S5000 functional layers

Connectivity layer
Virtually all standard H323 and/or SIP terminals can be connected to the S5000. This layer does not require any user actions. This layer implements H323, SIP, RSVP, TLS protocols.

Route and control layer
This is where rules apply for:

- a terminal/equipment to be accepted by the system
- a terminal/equipment to have a maximum bandwidth for its calls
- a terminal equipment to be accepted for a call
- a call to be directed to a location or another with or without numbers modifications

Routes and controls are set without any programming. These are simply set within the HTML graphical interface of the embedded web server or within the S5000 configuration file.

Supervision layer
Calls monitoring, endpoints information, instant bandwidths are monitored through associated views with the HTML web interface.

4. Installation

The S5000 Softswitch is multi platforms. The instructions below apply for most WINDOWS or LINUX platforms.
For any unlisted platforms, please contact our technical staff for certified installations or ports.

4.1. Prerequisites for S5000 execution

To run the S5000 software, please verify your platform satisfies the following:

- hardware requirements
- software requirements
- additional software requirements

4.1.1. Hardware requirements

Please note that other hardware may be supported or added. Please contact our technical staff.

Hardware	Requirement
Memory	128 MB of RAM is a minimum. 512 MB and more is recommended for best usage and performance.
Disk-Drives	A single disc drive is sufficient. 1 MB of Hard disk storage is required for the S5000 as a minimum installation. This amount does not include the OS and other necessary components.
Network cards	1 Ethernet network card with TCP-IP stack is a minimum. Any number of cards is supported.
CPU	Starting with Pentium II for light (up to 20 simultaneous calls) to Pentium IV 2Ghz and above for huge performance (200 calls and above)

4.1.2. Operating System requirements

OS software	Requirement
OS core	LINUX RedHat 6.2/7.3/9.0 LINUX Fedora Core 1/C2/C3/C4/C5/C6/C7 Linux RH Enterprise 3 WS Linux Ubuntu (8.04, 10.04), Debian or Microsoft WINDOWS 98, 2000, Me, NT4, XP, 2003, VISTA, Seven VM-Ware 5 WS, GSX server are supported.
OS Graphical environment	Not necessary. Web access can be used form local or remote platform with an HTML 3.2 Browser.

4.1.3. Additional software components requirements

Software	Requirement
Java Virtual Machine	Sun Microsystem, JRE 1.4 and above (a 50 MB disc drive is necessary) M2MSOFT's installers install the necessary JVM for you automatically. NOTE: An S5000 version for older systems with JRE Java 1.1 is available upon request. It runs on Kaffe Jvm1.0.6 and above (a 3 MB disc drive is necessary) and Sun Microsystem's JVM 1.1 and above. (TOS functions will be only activated with Sun Microsystem >= JRE 1.4.x)
M2Msoft ControlCenter	Mandatory for iPBX use
Directory	Mandatory for iPBX use. Use these for external Ldap storage of users. OpenLdap or Netscape Directory Server 4.5 and above
TFTP	Mandatory for iPBX use.
DHCP	Recommended for iPBX use.
NTP server	Mandatory for iPBX use.
MySQL server	Optional. Necessary for configuration redundancy in case of cluster use.
Web Browser	Optional. One can use the following Browsers : Microsoft Internet Explorer 4 and above Netscape Navigator 4 and above Konqueror, Mozilla FireFox, Apple Safari v3
M2Msoft License server (LI100)	Optional. Under conditions, S5000 license can be controlled by a remote network server, enabling the use of floating licences. Contact M2Msoft for more information.

4.2. The S5000 delivery package

The package is delivered on CD ROM or data file.

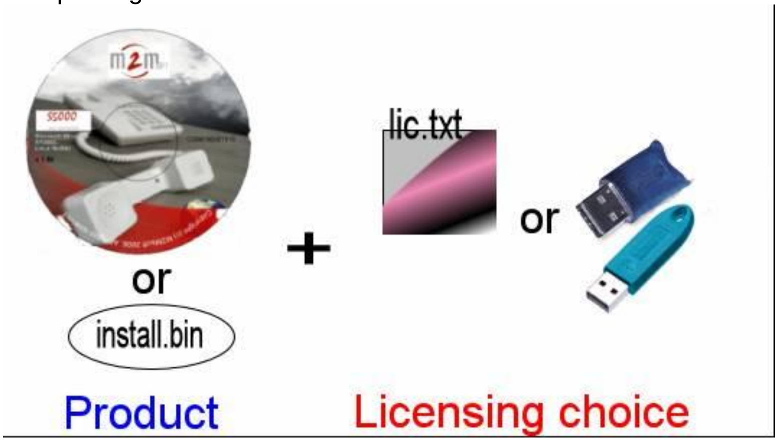


Fig.4 S5000 package

To run, the M2M-S5000 needs a license file or a USB dongle (or a license server under conditions). You need to contact your system administrator to get such file or dongle according to the features bought.
USB Dongles are unique to your delivery options and contains your specific licensing data.

There are automatic graphical installers for Microsoft Windows and Linux systems.
A manual installation for console only systems is explained in at the end of this part.
Licenses server use is described in 4.4.4 chapter. When using license server, you do not need any license file nor dongle on the S5000 station, but for the license server.

4.3. Installation

4.3.1. Start from CDROM installer

Your delivery package is made of a CDROM with an ***install.htm*** file.
Start your web browser and load the ***install.htm*** file, then, select your platform.

After clicking on your package, choose **Open** on memory on the dialog box that appears. This will execute the installer without saving unnecessary files on your hard drive.

4.3.2. Start from file installer

Your delivery package is made of a binary file **install.exe** (Microsoft Windows platform) or ***install.bin*** (Linux platform).
Start the installer as follow:

- Windows : double-click on install.exe
- Linux: open a shell and enter : sh ./install.bin

NOTE: This is a graphical installer, please start your X11, KDE, or Gnome environment on Linux systems before running the installer.

4.3.3. Graphical installation

Then the installer wizard will guide you through the installation steps.

A Java Virtual Machine is bundled and will automatically install locally with the product without conflicting with your other JVM if any. You will choose your hard drive directory for install and take a look at the release notes.

After the execution, you are done and you can start directly the S5000 program on Microsoft Windows by launching the ***gkmain.exe*** program or ***gkmain*** program on Linux system.

Startup scripts

Microsoft Windows	Linux	
gkmain.exe	gkmain	Default starter (no parameters)
s5kstart.bat	s5kstart.sh	Flexible starter that can be configured.

s5kstart.bat (Microsoft Windows) and s5kstart.sh (Linux)

4.3.4. Manual installation

This installation is for specific environments.
If you received a packaged file.
The package is composed of the file named such:

`s5000_1.95E-r1.tar.gz or s5000_1.95E-r1p1.tar.gz`

Note: The generic product filename is `s5000_x.y-rz.tar` where *x.y* is the product version and *z* the release value. (It can be used a patch with p in additional of r)

- Create a directory of your choice and install the package in this directory.

  NOTE: It is not necessary to be root or system administrator to install nor run the S5000 except for particular options enabling.

  The directory will be named <ROOT> in this document.

- Unpack the distribution file within <ROOT> directory.

  Windows users: use WinZip product

  Linux users: use tar command

```
# <ROOT> $ tar zxvf s5000_1.95E-r1.tar.gz
```

The directory content appears now like this:

<ROOT>		
	s5kj.jar	S5000 binary
	lic.txt	License file
	S5kimages.jar	Embedded web server pictures
	s5kstart.sh	S5000 start script (Unix version)
	s5kstart.bat	S5000 start script (Windows version)
	README.txt	Read this for information on this S5000 release
	License.txt	License contract
	<user>	Directory with gk.ini
	<user>/gk.ini	S5000 configuration file
	<media>	Directory with audio files
	<media>/sample_message_media.sw	Audio file as sample (G711Alaw format)
	<cert>	Certificates and private key directory
	<matrx>	USB Dongle management
	<tftp>	Directory with Auto-provisioning files

- Customize your starting script.

The s5kstart script must be customized according to your environment.
Edit the file to set some mandatory paths.
Windows users: use WordPad.
Linux users: use vi or emacs or usual graphical tool.

Here is the customization to be made:

Windows users	Edit line in s5kstart.bat	Modify with
	Set JAVA_PATH=<PATH_TO_JAVA_BINARY>	Enter here the complete path to the java binary. Example : set JAVA_PATH=E:\j2sdk1.4.2_01\bin
Linux Users	Edit line in s5kstart.sh	Modify with
	export JAVA_PATH=<PATH_TO_JAVA_BINARY>	Enter here the complete path to the java binary. Example : export JAVA_PATH=/opt/j2sdk1.4.2_02/bin

- Check you get a valid license file “lic.txt”
  The lic.txt file must be in the local directory as a default configuration. A parameter within gk.ini file allows changing this access.

- Start the S5000 binary

  Windows users: start the s5kstart.bat.

  The fig.6.below shows the execution starts. You may not have SIP system activated according to your license file but you do not have to get out of execution.

```
E:\S5000>E:\j2sdk1.4.2_01\bin\java -cp ./gims1.0.jar
JH323.Gk.gkmai
n -c E:\S5000\gk.ini
Configuration file 'E:\S5000\gk.ini'
Current call model : ROUTED
M2M-S5000 - Gatekeeper Module $Revision: 1.83 $ -
(c) 2003-2006, M2MSOFT - All rights reserved
 (-v for full copyrights) started on 193.7.1.213
License valid until :12/2006
gkcom ready for incoming connections
ready for incoming connection
Ready for incoming requests
SIP registrar server started
```

Fig.5 S5000 stdout starting traces

Linux users: `# <ROOT> $ ./s5kstart.sh`

The execution window looks the same as the fig.5.

The S5000 is now up and running, ready to register endpoints, gateways and conference bridges.

Please refer to your terminals and other equipment configuration manual to connect them to the Gatekeeper.
The Gatekeeper IP address must be the one you see on the starting window.
There is no filtering on endpoint registration in the default S5000 configuration.

4.4. License settings

After installing is done, one must set the license in order to have the S5000 run with the expected functions and capabilities.
The S5000 capabilities are controlled with a licensing procedure.
Licensing is based on either or the following modes:

- **Dongle system**, this allows for the maximum flexibility, the S5000 can be executed on different platforms with just the USB dongle to connect
- **License file system**, this does not need additional material and is the quickest way to run as we can deliver the file by electronic mail or http portal. This is also a way to limit program execution to only one hardware platform. The license file depends on the physical platform.
- **Licenses server system,** this is reserved to large environments with multiples M2Msoft products and stands for floating licenses. Your S5000 connects to a license server program on the network that gives or forbids the S5000 execution.

4.4.1. License general parameters

When you request an M2Msoft product license, either on dongle or file or license server (*), you receive the following data from M2Msoft by email or web portal or mail:

- option line
    - Mandatory data, prepared for you by M2MSoft that contain all your product capacities: maximum number of registered endpoints ('-t'), maximum number of concurrent calls ('-t' and '-k'), g729 capacity ('-g729'), etc. Ask your representative for details if needed.
    - All recognized capabilities are show within web page : "About": see "Installation checking" below
- description line

You need to keep these data for reference.
You need to enter these data in the *About* web page (see § 5.11.1), **exactly** as shown on the documents from M2Msoft.

(*) Only available under conditions. Contact M2Msoft if you need a license server licensing

The web access is allowed even with no valid license.

If you have a limited time license, this will be shown automatically at run time and within *About* web page (see *Installation checking* § 4.5, and *About* web page § 5.11).

4.4.2. Dongle settings

Enter the following in the 'About' license parameters (web page § 5.11.1).

Description	dongle
License options	-t 200 --sip -me 099 -rsvp

← ***Provided options example***

Please note:

The dongle you received is hard-coded for you, this is not an USB storage key and is dedicated to be used with your S5000. **You do not need to install any additional drivers after a correct S5000 installation**, the dongle will be detected automatically by the S5000 product.
Be sure to let the dongle connected all the time running the product.
Note: for Linux kernels v2.6 and above please see the appendix 8 for udev configuration.

4.4.3. License file

Enter the following in the 'About' license parameters (web page § 5.11.1).

Description	GSSTelecom
License options	-t 200 --sip -me 099

← ***Provided options example***

Copy your license ***lic.txt*** file within the installation directory of S5000 or enter the license key (32 chars) directly from this next field.

License key	:	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

4.4.4. License given from license server

Enter the following in the 'About' license parameters (web page § 5.11.1).

Description	:	GSS Telecom
License options	:	-t 200 –sip -me 099 -group -rsvp -pt120
License key	:	xxxxxxxx
License servers	:	192.168.0.111:56002

← ***Provided options example***

You do not need any ***lic.txt*** file within the installation directory of S5000.

The important field is the License servers' field that keeps one or more license servers.
Enter here ip address and tcp listen port of the M2Msoft license servers available on your network.

License servers	:	192.168.0.111:56002

The M2Msoft license server can distribute any number of licenses and is named LI100. The LI100 is available under conditions.

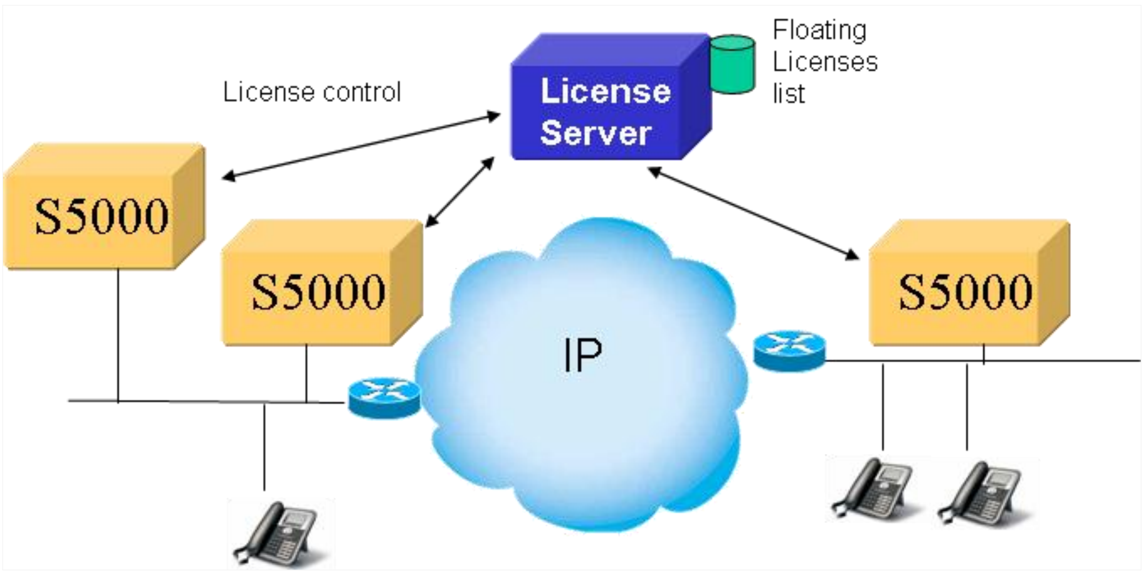


Fig.6 S5000 connected to remote license server

4.5. *Installation checking*

In order to check the installation, you first have to start the software (see previous chapter). Then, take advantage of the embedded web interface (see§ 5).

- Start a web browser on the local host or on a remote machine.
- Choose the following URL (port can be configured, the default is shown here):

http://<s5000-IpAddress>:8000

You have to check if the license is valid.
If the license is not valid or not found, **NO LICENSE** will be displayed on each web page.
You can verify the license options and some other information about product by clicking the “About” button:

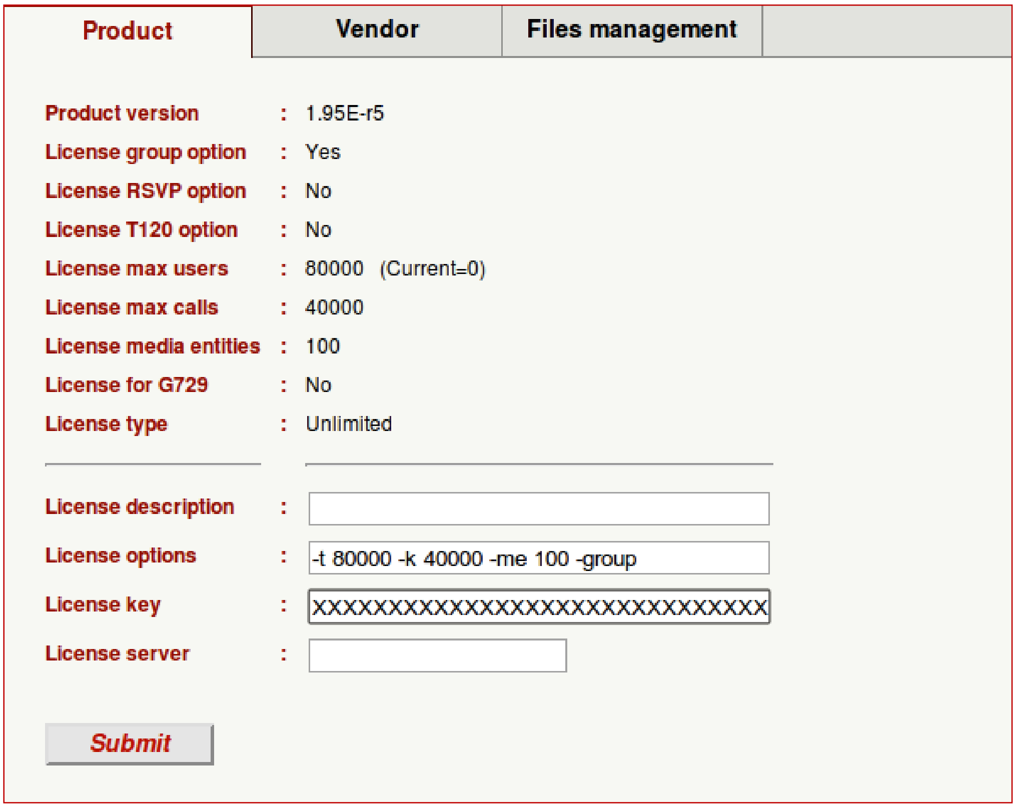


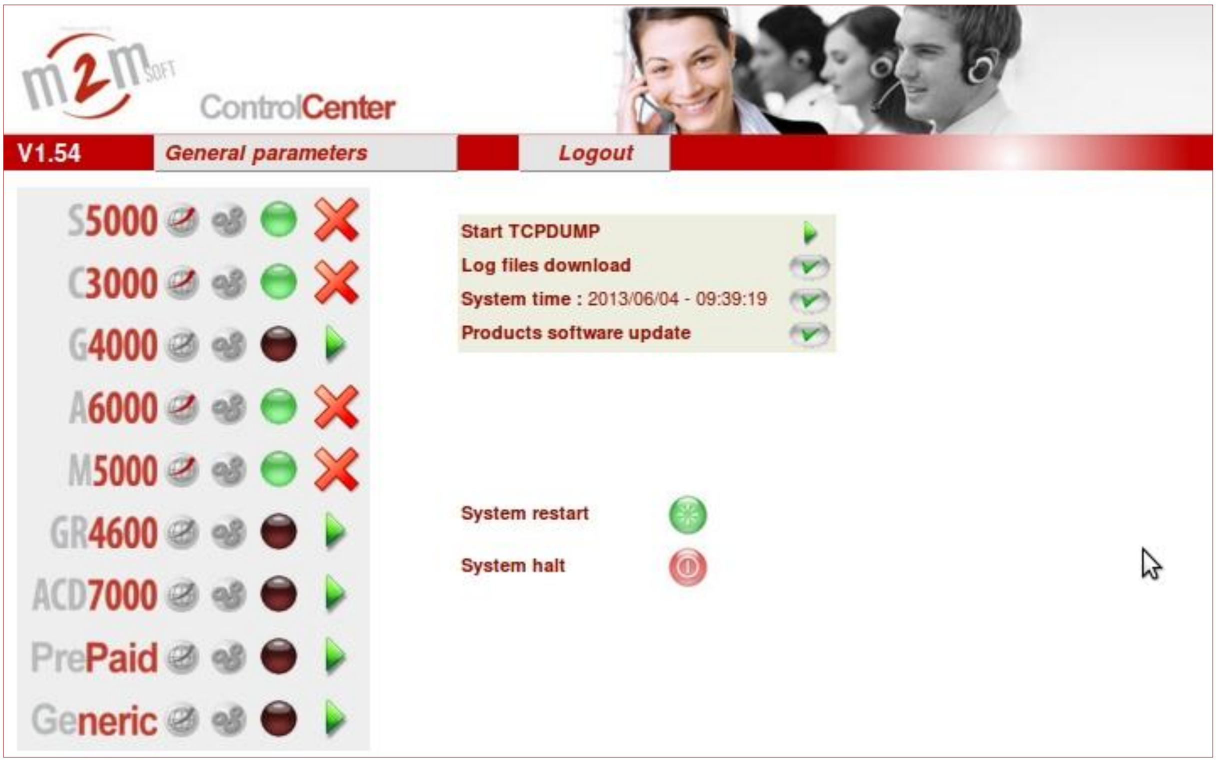
Fig.7 S5000 About page

Troubleshooting:
In case of an administration page unavailable, please check the following:

- *check your IP connection to the host*
- *check that the s5kstart script is still running (or check java processes on the platform)*

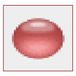
4.6. ControlCenter

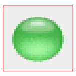
The M2Msoft ControlCenter acts as a watchdog monitoring selected M2Msoft applications such as S5000 and some other M2Msoft products.
The ControlCenter must be installed on a Linux Debian/Ubuntu platform.
It allows to configure the Ethernet interface parameters.
To access to the ControlCenter Web: http://<ipaddress>
The default M2Mbed device IP address is 192.168.3.20.
The default ControlCenter administrator password is ‘admin’.



Access to the application web service (if running).


The application is stopped and not monitored.


The application is monitored but is not running.


The application is running.


Access to application monitoring parameters.


Start the application.


Stop the application.

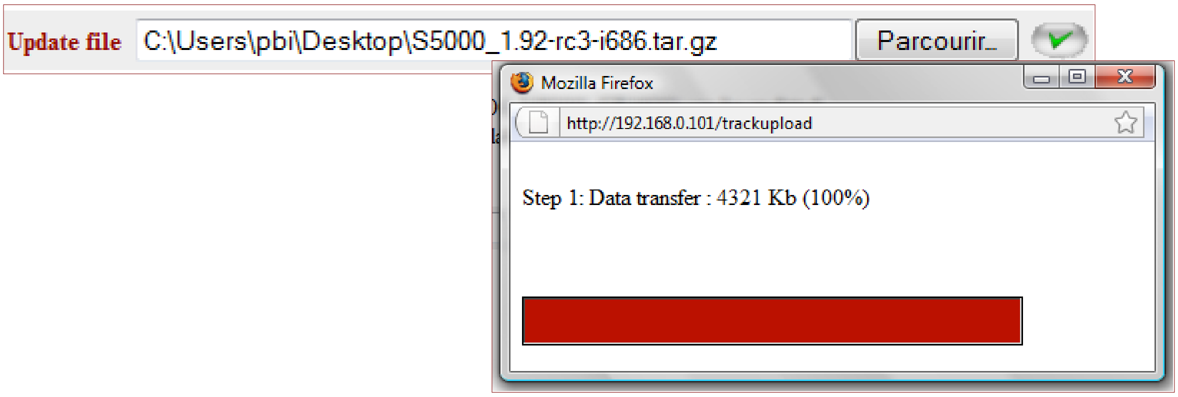

Halt the device.


Restart the device.

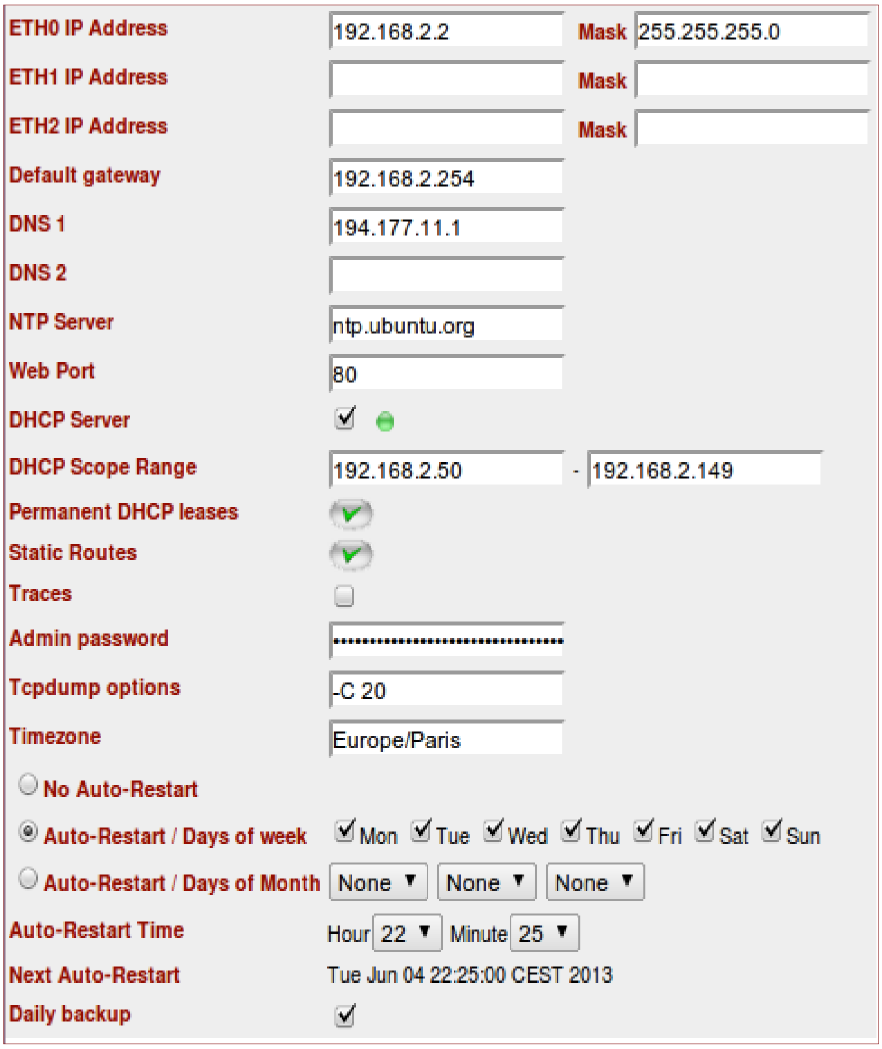
General parameters Access to ControlCenter general parameters.

Submit Save all ControlCenter settings.

- *Start TCPDUMP* Start/Stop a TCPDUMP capture on network interface.
- *Log files download* Access to the download page (Tcpdump and logs files).
- *System Time* Update the system Date and Time.
- *Products sw. update* To update any of products software



General Parameters:

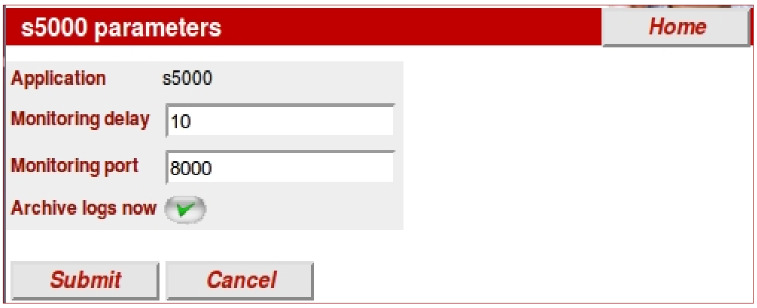


ETH0 IP address	IP address for first Ethernet interface.
ETH0 Mask	IP mask for first Ethernet interface.
ETH1 IP address	IP address for second Ethernet interface.
ETH1 Mask	IP mask for second Ethernet interface.
ETH2 IP address	IP address for third Ethernet interface.
ETH2 Mask	IP mask for third Ethernet interface.
Default gateway	Default gateway for first Ethernet interface.
DNS 1	First DNS server IP address.
DNS 2	Second DNS server IP address.
Web port	ControlCenter HTTP port (default=80).
DHCP Server	Enable/Disable DHCP server. (Runs only if scope is correct).
DHCP Scope Range	IP addresses range for phones and PC (within local subnet).
Perm. DHCP lease	To set IP/MacAddr mappings
Static Routes	To set IP static routings.
Traces	To run ControlCenter logs.
Admin password	Web ControlCenter password.
Tcpdump options	Tcpdump maximum file size.
Timezone	
Auto-Restart param.	Scheduler for automatic system restart.
Daily backup	When checked a complete configuration backup is done at restart time.

Application monitoring parameters:

This section allows to configure:

- The application monitoring status
- The monitoring delay
- The web port of the monitored application
- A command to archive product logs

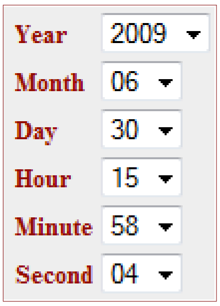


In the following case the s5000 is monitored by checking response on port 8000 every 10 seconds.

- *Monitoring delay* Delay in seconds between application checks
- *Monitoring port* TCP port used for application check (web port)
- *Archive logs now* To store and archive application logs.

System time update:

This section allows to locally update date and time (necessary if NTP requests over Internet are not allowed)



Log files download:

5. Configuration and Administration

The S5000 provides an embedded Web interface which let you configure and supervise the system.
The web interface has an opened and a secured mode with login/password and accounts.
If the secured mode is enabled we first have to login to access (see § 5.3.8).
The web interface has also a crypt mode with TLS access through https.

`http://<S5000 ip address>:<configured port>`
or
`https://<S5000 ip address>:<configured port>`

5.1. Login Page

NOTE: By default, the secured mode (Multi-Users) is disabled and no default user is configured.

5.2. Home Page and persistent buttons

At the left side of each page a list of buttons is persistent. It lets you directly access from any page to another topic.

f

Fig.8 S5000 letf side buttons

In secured mode all pages contain the button Logout to go back to the Login page:

Logout

All the pages (excepted Home and Login pages) contain the button Home to go back to the Home page:

Home

5.3. General parameters page

General parameters page is subdivided in several sections selectable with tabs.

General	SIP	H.323	Groups	dBase	Security	Http Accounts

The following sub-chapters describe all the parameters definition.

5.3.1. General

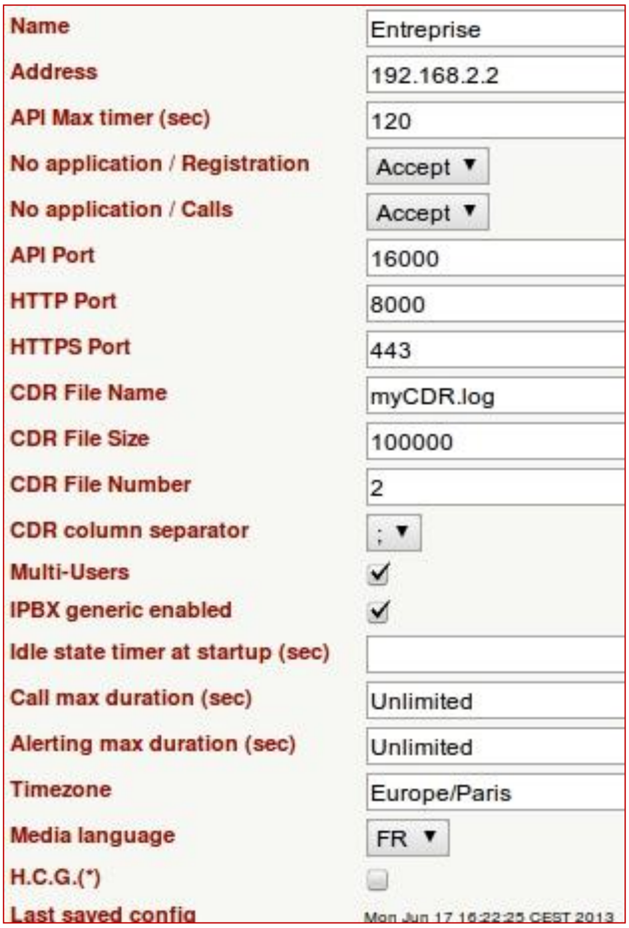


Fig.9 S5000 General Parameter page

- *Name* — A softswitch name to be displayed in the Web-based administrative interface. (Useful when multiple S5000 are started).
- *Address* — IP address for the Gatekeeper. * means listen on all interfaces. If an address is set, only calls received on the associated network interface will be handled.
- *API Max timer* — Timeout after a request sent to an application without response (in sec).
- *No API / Registration*
- *No API / Calls*
- *API Port* — The TCP port for the Application Program Interface (API). This is the port number that the applications will use to reach the S5000.
- *HTTP Port* — Defines the web portal access port.
- *HTTPS Port* — Defines the secured (HTTPS) web portal access port (0 disables it)
- *CDR File Name* — Set a CDR file name to be written in real time. Files are names

<CDRfile>_date.extension where extension is 0, 1,...

- *CDR File Size* Global size of all CDR files, in bytes. System is overwriting then the cycling file system.
- *CDR File Number* Maximum number of CDR files for each date.
- *CDR column separator* CSV file columns separated with character ';' or ','
- *Multi-Users* When checked the web is secured by HTTP accounts. (See § 5.3.8).
- *IPBX generic enabled* Enable/Disable enterprise IPBX features (If IPBX is enabled a local MTP must be used for all calls, see Media chapter).
- *Idle state timer at startup*
- *Call max duration* For SIP-SIP calls, automatically clear call after that duration in seconds. Set -1 to have unlimited duration.
- *Alerting max duration*
- *Timezone*
- *Media Language*
- *H.C.G.*
- *Last saved config.* Date of configuration file in use.

5.3.2. SIP

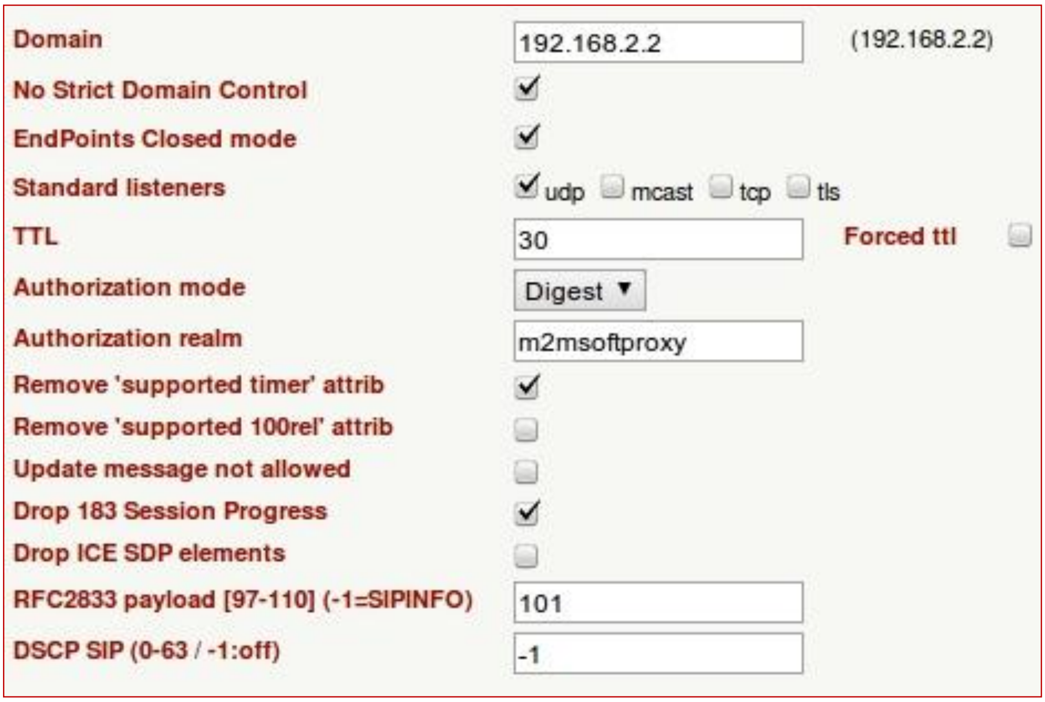


Fig.10 S5000 SIP parameter page

- *SIP Domain* The SIP default domain for the S5000 that acts as REGISTRAR and PROXY server. In case a SIP terminal does not provide any domain (host or IP address), this domain will be set by default. For the SIP to H323 communication, all H323 systems are added an H323 alias in the form e164@defaultDomain.
  Note: if an invalid value is set there, calls may not establish correctly.

- *No Strict Domain Ctrl* If checked, allow endpoints that come with different requested domain, to register here.

- *EndPoints Closed M.* ***Intelligent Security System.*** If checked, make a strict check of all registrations and calls based on what is known at time of parameter check.
  -Only already registered terminals can register again and back.
  -Anti spoofing is made on those terminals to enhance protection
  -Only known external IPs can have calls entering
  -Only registered endpoints can have outgoing calls
  This mode can work alone or coupled with the DIGEST mode to have a more protected system.
  - Intrusions are logged within Logs/Intrusion page
  NOTE: however a fully protected system will have to add protection on the router and firewall rules and monitoring (these are administrator tasks)

- *Standard listeners* Transports layers can be enabled/disabled here. Choose any of:
  udp: SIP UDP unicast listener. UDP port=5060.
  mcast: SIP UDP discovery multicast listener. UDP port 5070.
  tcp: SIP TCP listener. TCP port=5060.
  tls: SIP TLS listener. TCP port=5061. TLS layer needs some security settings to work. *See § 6.10 for detailed information about S5000* Secured Transport Layer feature.
- *SIP TTL* The time to live, in seconds, to apply to endpoints that do not advertise any duration. It is also the minimum TTL accepted for registrations.

- *Forced TTL* This is the forced ttl replied to registrations. Checking this will force endpoints to adjust their ttl to the SIP TTL value.

- *SIP Authorization* 'None' when no special registration control is done, and 'Digest' (RFC2617), when a challenge operation will take place at every registration to control the endpoint name and password. The password is to be set within the Endpoint profile access.

- *SIP Authorizat. realm* This is the special name associated with a SIP Digest account. Used when mode=DIGEST.
  A special realm value is sometimes expected from SIP endpoints or iPBX. Change it to your needs. (RFC2617).

- *Remove support timer* When checked s5000 does not forward the "timer" phrase from "Supported" attribute of INVITE SIP messages. It is useful to disable the Session expiration timer of SIP endpoints.

- *Remove support100Rel* When checked s5000 does not forward the "100Rel" phrase from "Supported" attribute of INVITE SIP messages.

- *Update message not...Avoid the UPDATE message being supported in calls.*
  Some systems use the UPDATE as a polling purpose and this may be forbidden here.

- *Drop 183 Session Prog* When checked s5000 discard the 183/Progress message which can prevent to hear the Early Ringing even

- *Drop ICE SDP elements*

- *RFC2833 payload* Value (97→110) for RFC2833 DTMF according value in IPBX global provisioning. See § 5.8.4. (-1 in case of SIP-INFO DTMF).

- *DSCP SIP (0-63)* *Force the DSCP field to be set on outgoing SIP signalling messages.*

5.3.3. H.323

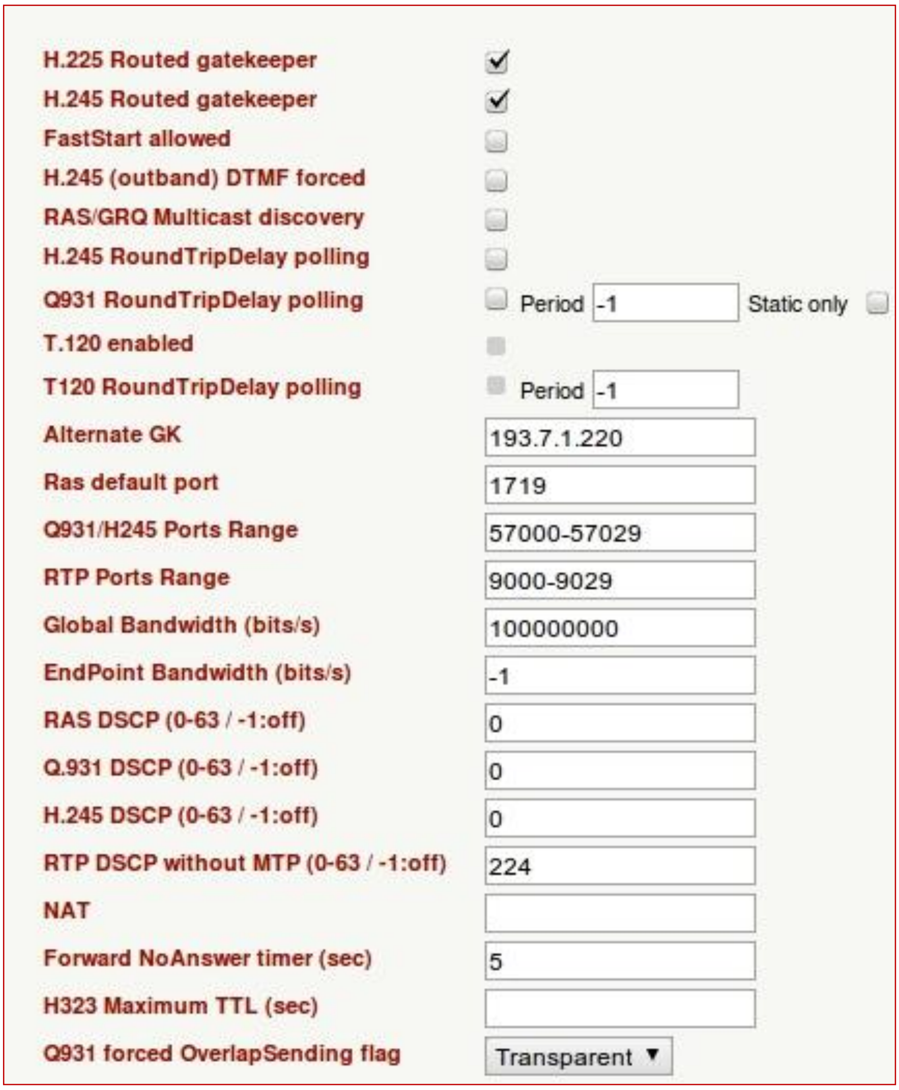


Fig.11 S5000 H323 parameter page

- *H.225 Routed* When checked, the H225/Q931 messages are controlled by S5000. Else S5000 process only RAS messages.
- *H.245 Routed* When checked, the H245 messages are controlled by S5000.
- *FastStart allowed* When not checked, the S5000 prevents FastStart mode on all H323 calls.
- *H.245 DTMF forced* When checked, force H245 DTMF capability to be out of band only.
- *RAS/GRQ Multicast* Enable/Disable RAS multicast listener for Gatekeeper discovery.
- *H245RoundTripDelay* Enable/Disable H245 endpoints polling for keep alive while in call.
- *Q931RoundTripDelay* Enable/Disable Q931 endpoints polling for keep alive while in call.
- *T120 Enabled* Enable/Disable T120 proxification on this platform. If licensing does not allow this, the button is disabled.
- *T120RoundTripDelay* Enable/Disable T120 endpoints polling for keep alive while in call.

- *Alternate GK* Alternative H323 Gatekeeper IP address or hostname. *See § 6.8 for detailed information about Resilient Solution.*

- *Ras Default Port*

- *Q931/H245 Ports Rnge* Q931 and H245 TCP ports range. Define the first and last allocated ports within the S5000. This feature is very useful for secured network in which only selected ports are allowed.

- *RTP Ports Range* RTP UDP ports range. Define the first and last allocated ports within S5000 for media Entities or RTP translations. Each call needs 2 ports (RTP and RTCP sessions). This feature is very useful for secured network in which only selected ports are allowed.

- *Global Bandwidth* Global bandwidth (in bits/sec) available at start for all calls.
-
- *EndPoint Bandwidth* Global bandwidth value (in bits/sec) per endpoint. This is the maximum allowed per endpoint per default. (An endpoint block can change the maximum terminal value for a particular endpoint). When is set to -1 no bandwidth is allocated to endpoints.

- *RAS DSCP*
- *Q.931 DSCP*
- *H.245 DSCP*

- *RTP DSCP*

- *NAT* NAT IP address for all calls. (Optional).

- *FwdNoAnswer timer* Timeout to forward call in No Answer case (in sec).

- *H323 max TTL* Maximum TTL accepted for endpoint registrations.

- *Q931 forced Overlap*

H.323 Zones

These are the rules defined for H323/RAS LRQ (Location Request) messages: adjacent gatekeepers can be defined here.
But we recommend to use an alternate and generic way, suitable to all environments, to route calls towards remote S5000: Static Entity + Routes in Embedded Services.

- *Name* Zone name.
- *Destination Mask* Destination pattern to reach endpoint on remote gatekeeper.
- *Remote IP* Remote gatekeeper IP address (default RAS port=1719).

H.323 Proxies

H323 Proxies are used to connect to external Gatekeepers that do not support calls to unregistered endpoints.
H323 Proxy defines an entity within the S5000 that registers as an endpoint to an external gatekeeper. Calls to this entity within the external gatekeeper are automatically handled within the S5000 and then routed to any recipient, registered or not.

Name	E164	IP address	
proxyParis	1234	192.168.0.50	
proxyLondon	5678	195.234.88.2	

Name	proxyLondon
E164 alias	5678
H323 alias	proxy1
IP address	195.234.88.2

- *Name* Proxy name.
- *E164 alias* E164 alias for gatekeeper registration.
- *Remote IP* H323 alias for gatekeeper registration
- *IP address* Gatekeeper IP address

5.3.4. Bandwidth areas

The S5000 allows for multiples WAN links controls within a single S5000 controller entity.
It is not sufficient to limit globally the bandwidth; a more accurate view can be achieved when the managed network is split as several areas with low bandwidth inter-areas.
By applying intelligent numbering plan and inter-areas bandwidths, the S5000 can handle the maximum allocated bandwidth on all its managed endpoints.
To avoid congestion, and not apply a global bandwidth limit, one can define rules per route (source→destination).

Name	Source Mask	Destination Mask	Bandwidth	Available Bw	
BW_Toulouse_London	000*	111*	256000	256000	
BW_Toulouse_Tokio	000*	222*	128000	128000	

Name	BW_Toulouse_Tokyo
Source Mask	000*
Destination Mask	222*
Bandwidth	128000

- *Name* Bandwidth area name.
- *Source Mask* Pattern matching calling party number.
- *Destination Mask* Pattern matching called party number.
- *Bandwidth* Allocated bandwidth in bits/sec.

5.3.5. Groups

Only available with Group option. See § 6.8 for detailed information about Resilient Solution.

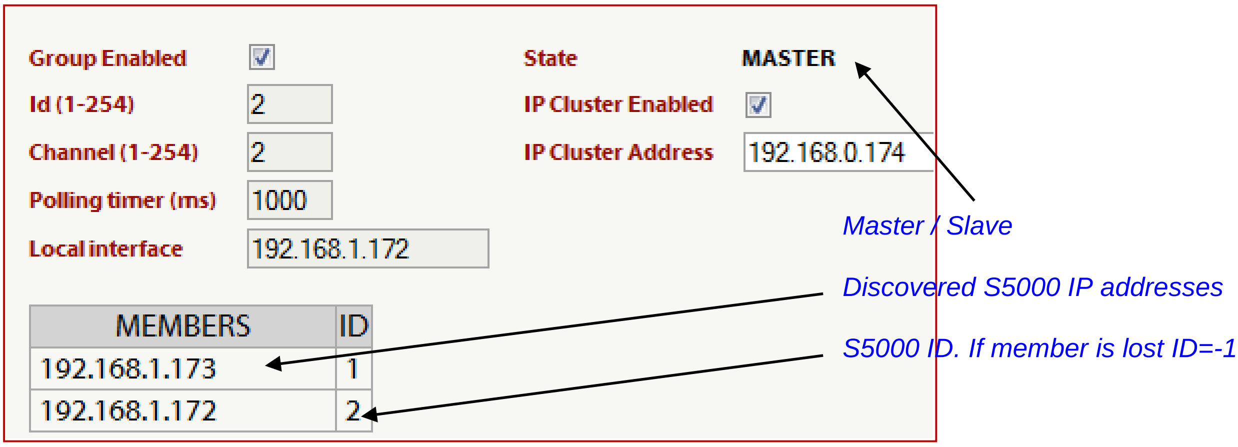
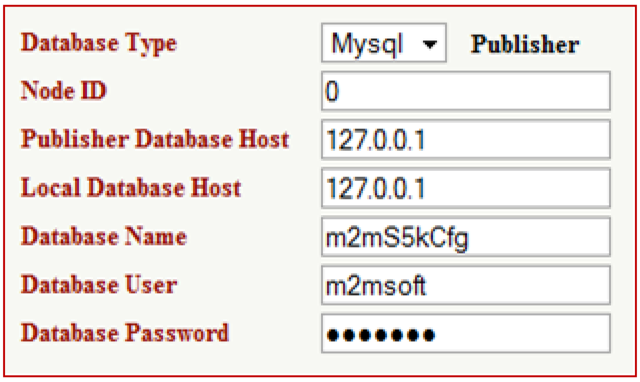


Fig.12 S5000 Groups parameter page

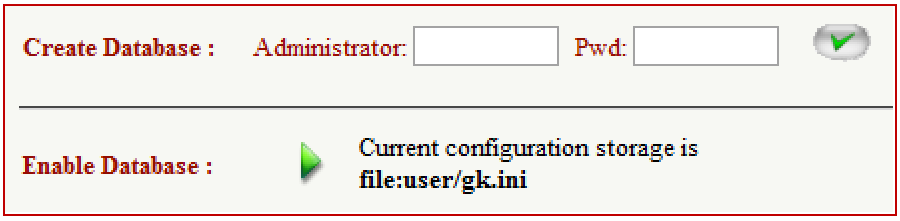
- *Group Enabled* Activate the Group mechanism (must be disabled to change parameters).
- *Id* S5000 identifier in a group. The Master is the S5000 with highest Id.
- *Channel* Group identifier.
- *Polling timer* Define a polling period for every S5000 within the group to advertise the others (ms).
- *Local interface* To force a local interface to transmit and receive the supervision messages (useful when several Ethernet interfaces) .
- *IP Cluster Enabled* Activate the Cluster option which provide a virtual IP address as a unique address for all group members
- *IP Cluster Address* Virtual IP address for cluster.

5.3.6. DBase

This allows to store the S5000 configuration within database which can provide replication between a publisher and subscribers server, useful in case of group running.



- *Database type* Only MySQL available.
- *Node ID* Each node must have an unique ID
- *Publisher db host* Publisher IP database address
- *Local db host* IP database address for local S5000 (same as previous if publisher)
- *Database Name* Database name for S5000 configuration storage
- *Database User* Database username for queries authentication.
- *Database Password* Database password for queries authentication.



A wizard is available to create a local database with previous parameters. The administrator account is required to do this.
To enable database storage click green arrow. All S5000 configuration parameters are converted from user/gk.ini file toward database.
To disable database storage click Red Cross. All S5000 configuration parameters are converted from database toward user/gk.ini file.

5.3.7. Security

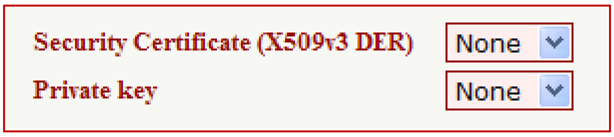
See § 6.10 for detailed information about S5000 Secured Transport Layer feature.

The S5000 can use TLS secured links for **HTTPS** access and **VoIP** signaling.

Please note:

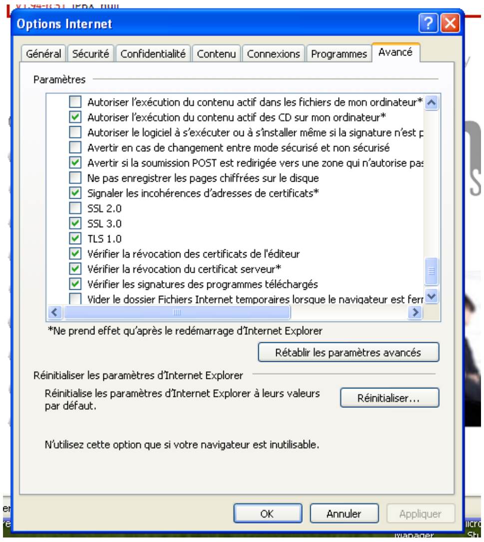
Restriction may apply for exportation. Special binaries with low or no cryptography are available. Please contact us.

For TLS/HTTPS and VoIP TLS, two (administrator defined) files must be entered.



- *Security Certificate* X509v3 asn1 DER certificate file.
- *Private key* Key file in PKC#8 format, asn1 DER.

Https browser compatibility

Browser	OS	Note
Mozilla FIREFOX 3.5.13	Linux UBUNTU 8.04 Microsoft Windows XP	
Microsoft IE 8.0.6	Microsoft Windows XP	Please force SSLV3 
Apple SAFARI 5.0.5	Mac OS X	

5.3.8. HTTP accounts

The web interface can be secured with definition of login/password accounts that are authorized to connect. In this mode, only defined users can connect and see the S5000 administration and configuration. Sessions have limited time duration (15 minutes of non-use).
There are different levels of accounts, with special properties.

The mode is enabled/disabled within the "General Parameters" see § 5.3.1).

Name	Login	Level	Language	LAST LOGIN	
admin	admin	0	EN	Tue Jun 18 2013 09:54:00.059	
fred	fred	1	FR	-	

- *Name* Account name.
- *Login* User identifier.
- *Password* User password.
- *Level* 0=Admin level: All data can be read and written, reset can be done, save can be done.
  1=User level: All data except http account and Multi user mode can be read and written. Reset: disabled; save: disabled
  2=Guest level: Data are READ ONLY. Reset: disabled; save: disabled. Nothing can be modified. Useful for DEMO access.
  3=level:
- *Language* Web User language.

***NOTE**: It is also possible to access to an IP-Phone settings in user mode: Login is #<extension> (see § 5.4.4).*

5.4. *Endpoints page*

Endpoints page is subdivided in several sections selectable with tabs.

Endpoints	Endpoints Profiles	Static Entities	SIP Accounts

This page allows to supervise endpoints, affect parameters to endpoints, create static (non-registered) endpoints and manage SIP registration accounts.
The following sub-chapters describe all the features.

5.4.1. Endpoints view

Filter [ ] Lines/page [50]

Alias ▼▲	Sig. ▼▲	Type ▼▲	Address ▼▲	TTL ▼▲	Grp ▼▲	Superv ▼▲	
5100 (5100)	SIP	REGISTERED	192.168.3.18:5060	60			unregister
5179 (Standard)	SIP	REGISTERED	192.168.3.41:5060	60	Group		unregister
5188	H323	REGISTERED	192.168.3.39:1720	300			unregister
52	H323	REGISTERED	192.168.3.36:1720	60			unregister
54	H323	REGISTERED	192.168.3.35:1720	90			unregister
6000 (a6000)	SIP	REGISTERED	192.168.3.31:38070	63			unregister
911		MEDIA					connectTo
912		MEDIA					connectTo
914		MEDIA					connectTo
99001		MEDIA					connectTo
9908		MEDIA					connectTo
SIPLISTENER5100	SIP	STATIC	192.168.3.31:5100				
lab30	H323	STATIC	192.168.3.31:18000				
sip:listener_TCP	SIP	STATIC	192.168.3.31:5060				
sip:listener_TLS	SIP	STATIC	192.168.3.31:5061				
sip:listener_UDP	SIP	STATIC	192.168.3.31:5060				
sip:listener_UDP	SIP	STATIC	192.168.3.31:5070				

Fig.13 S5000 Endpoints view page

It is possible to sort this list by Alias, by signalling type (H323 or SIP), by Type, by address, or by TTL value: use the arrows ▼▲ in the title columns.
It is also possible to restrict the list using a filter example:

Filter [SIPREGISTERED] Lines/page [50]

Alias ▼▲	Sig. ▼▲	Type ▼▲	Address ▼▲	TTL ▼▲	Grp ▼▲	Superv ▼▲	
5100 (5100)	SIP	REGISTERED	192.168.3.18:5060	60			unregister
5179 (Standard)	SIP	REGISTERED	192.168.3.41:5060	60	Group		unregister
6000 (a6000)	SIP	REGISTERED	192.168.3.31:38070	63			unregister

Fig.14 S5000 Filtered Endpoints view page

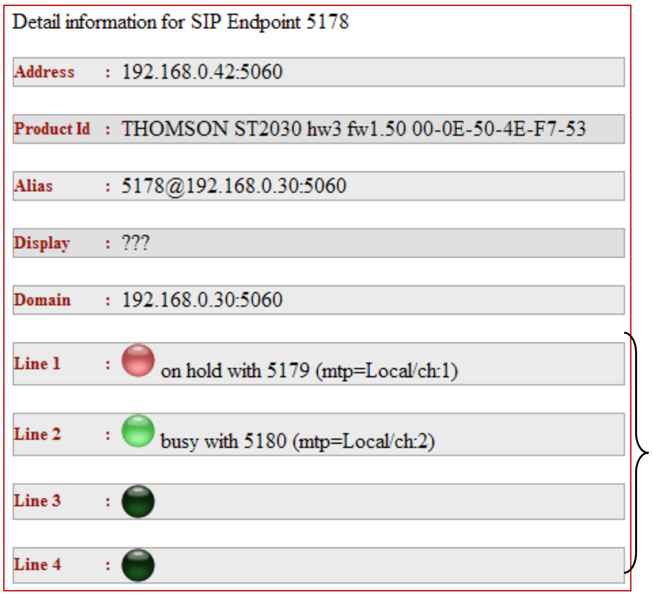
This area displays 3 types of endpoints:

- Registered endpoints,
- Static endpoints,
- Media entities.

A registered endpoint is a terminal, or MCU, or gateway, or proxy which dynamically registers to S5000 using H323/RAS or SIP protocols.
This type of endpoints is displayed with:

- Main E164 alias
- H323 or SIP signalling protocol
- Signalling address
- TTL (Time To Live) value.
- A command to “unregister” this endpoint.

By clicking on Alias (shown as a link) a child window displays detailed information (Product ID, extended aliases…) about the endpoint:



Virtual lines status in IPBX use

A static endpoint is a static definition of a remote entity or a local listener.
This type of endpoints is displayed with:

- Static entity name
- H323 or SIP signalling protocol
- Signalling address

To create a static entity see the § 5.4.5

A media entity is an embedded media terminal (see § 6.2 for more information about media entities).
Only its name (as an alias) configured in Media Entities chapter (see § 5.7.1) is displayed.
The link “***connectTo***” allows dialOut call to a remote endpoint (see Media Entities § 6.2).

5.4.2. Endpoints Profiles / Basic parameters

This is used to define “a priori” **redirection** numbers for registered users.
Audio RTP port can be set for H323 to SIP calls to have direct media routing between endpoints. The known RTP port for the H323 endpoint can be entered here. When set, this number is advertised to called peer SIP endpoint in sip INVITE signaling.
In H323 to H323 mode through **NATted systems**, this can be used to have bidirectional RTP flow (listener and receiver on the same source port viewed from the external sites.) Used in call with destination is a StaticEntity with RTP enable and NAT address set.
Password can be set and used when S5000 SIP Authorization mode is set to “Digest”, to challenge registrations from endpoints.

NOTE: Embedded Services have a higher priority over these rules.

Name	Alias	Auto-provision	Type	Grp	
6000	6000	None	-		
52	52	None	-		
5176	5176	001F9F1641F8	Thomson-ST2030	Group	
5183	5183	00264430AE5F	Thomson-TB30	Group	
5199	5199	0080F0C8B253	Panasonic-UT136		
5119	5119	00085D2F6006	Aastra-6757i		
5120	5120	00085D301039	Aastra-6731i		

Name	5120	Lost Registrations 0.0% (0)	
Alias	5120	LEARNING	
Web user password	••••		
NAT			
Audio RTP			
Group			
Gateway mode	☐		
Allow re RRQ with diff. address	☐		
H323/SIP with SDP in ACK	☐		
URI alias as Contact	☐		
Sip call supervised	☐		
Registration linked with	None		
Dialout Restrictions	None		
SipCallingPrefix for xfer			
Forward type	None		
Forward to messaging	☐		
Forward to other destination			
Forward NoAnswer timer (sec)	10		
Shared line			
EavesDropping allowed	☐		
EavesDropping mask			
Codecs filtering			
Forced G729	None		
Special codec 1		ptime 1	None
Special codec 2		ptime 2	None

Fig.15 endpoint profile definition

- *Name* Profile name.
- *Alias* Endpoint alias associated to the profile.
- *Web User Pwd*
- *NAT* NAT IP address for this endpoint (empty if no NAT).
  The NAT address can be automatically detected when the endpoint registers from a private network to a S5000 in a public network. In this case no entry is needed. For endpoints that registers from internet into a private network S500, NAT setting is needed here with public S5000 address
- *Audio RTP* Set H323 endpoint RTP port. Used for calls toward SIP endpoints.
- *Group* Group name for call Interception feature (IPBX use).
- *Gateway mode* To replace the endpoint alias in SIP INVITE request-URI field by the real destination (copied from TO field).
- *Allow Re RRQ*
- *H323/SIP with SDP*
- *URI alias*
- *Sip call supervised* To check if the endpoint is still alive during a call.
  (Useful to manage calls when the Wi-Fi phones loose the radio cover during the communication).
- *Registration linked with* To accept this endpoint registration only if a ClientRegistrar/StaticEntity is already registered (SIP operator). See § 5.4.5.
- *Dialout Restrictions* To apply a call restriction defined at § 5.6.6.
- *SIP CallingPrefix*
- *Forward Type* Destination alias to forward None, Always or Busy/No answer calls.
- *Forward to Messaging* Destination alias to forward to Messaging.
- *Forward to other dest* Destination alias to forward to other destination.
- *Forward to No answer* Timer to forward calls.
- *Shared Line*
- *Eaves Dropping Allow*
- *Eaves Dropping Mask*
- *Codecs filtering*
- *Forced G729* Packetization
- *Special Codec*
- *Ptime*

5.4.3. Endpoints Profiles / Auto-provisioning

Manual configuration:

In IPBX mode some IP-Phones configurations are managed from S5000.
When auto-provisioning is enabled the phone configuration is provided to the device by TFTP.
For a large IP-Phone deployment, a serial auto-provision can be built from a CSV file (see ‘CSV file transfers’ paragraph later in this chapter).

Name	Alias	Auto-provision	Type	Grp	
6000	6000	None	-		
52	52	None	-		
5176	5176	001F9F1641F8	Thomson-ST2030	Group	
5183	5183	00264430AE5F	Thomson-TB30	Group	
5199	5199	0080F0C8B253	Panasonic-UT136		
5119	5119	00085D2F6006	Aastra-6757i		
5120	5120	00085D301039	Aastra-6731i		

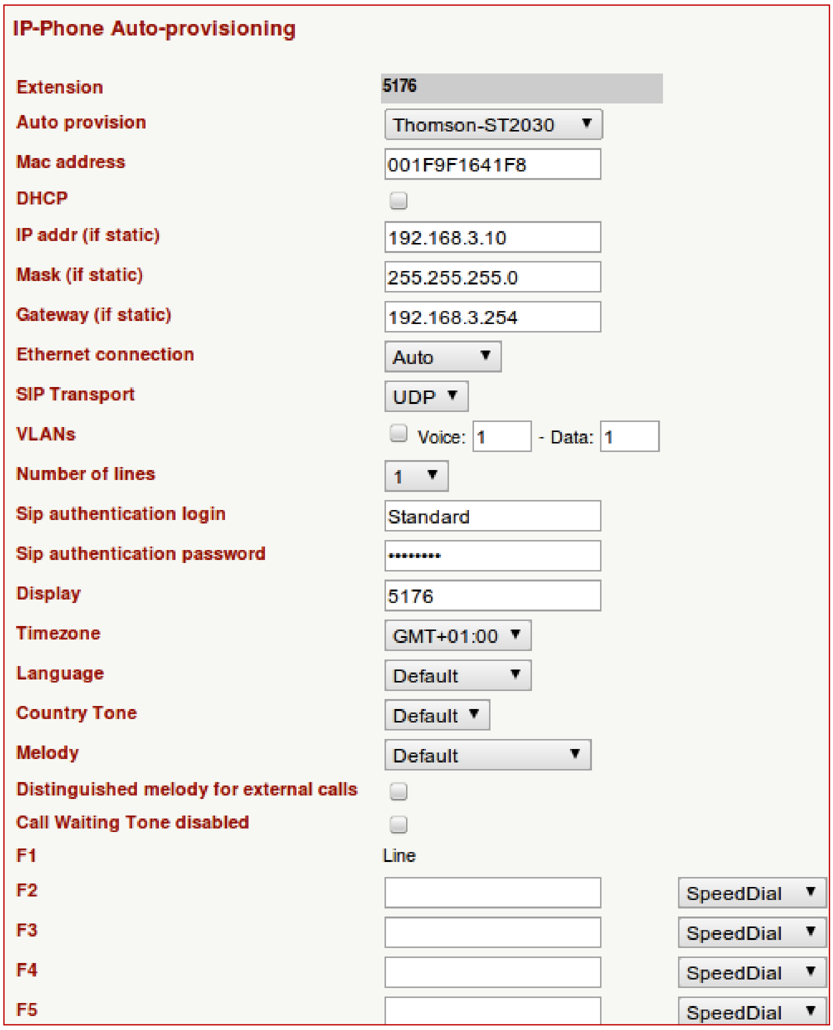
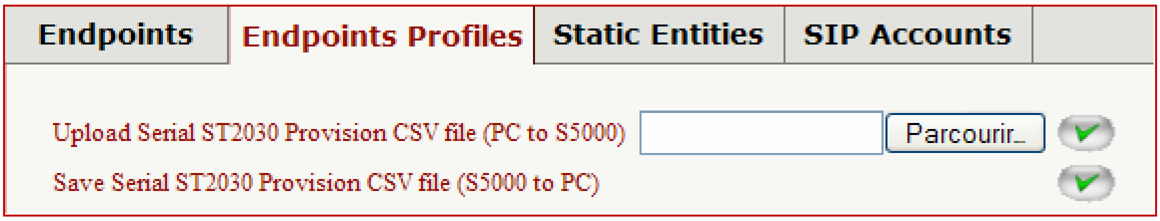


Fig.15bis IPPhone auto-provisioning

- *Extension* IP-Phone number.
- *Auto provision* Enabled if IP-Phone type is selected (Thomson, Panasonic or Aastra).
- *Mac address* Device MAC address.
- *DHCP* Dynamic IP address: Preferred configuration to obtain automatically the TFTP server address.
  In case of static IP address at least the TFTP Server IP address and boot file must be configured within IPPhone.
- *IP addr* Device IP address if DHCP disabled.
- *Mask* Device IP subnet mask if DHCP disabled.
- *Gateway* Default IP gateway if DHCP disabled.
- *Ethernet connection* Auto, 100Mbps/Half-duplex, 100/Full, 10/Half, 10/Full.
- *SIP Transport* UDP or TCP Transport
- *VLANs* Distinguished Vlan for Voice and Data.
- *Number of lines* Max concurrent multiline (1~10).
- *SIP authentication login* SIP phone login required if DIGEST mode is enabled (see § 5.3.2).
- *Sip authentic. password* SIP phone password for DIGEST mode
- *Display* User identifier displayed on called phones.
- *Time Zone* IP-Phone local time zone (Default=value in IPBX page).
- *Language* IP Phone language displays. (Default=value in IPBX page).
- *Country Tone* IP Phone country for tones. (Default=value in IPBX page).
- *Melody* IP Phone ring melody.
- *Disting. Melody for ext...* If checked, the selected melody will be modified for calls from outside.
- *Call waiting tone disabled* To prevent tone when a call comes on 2nd line.
- *F1 ~F10* Device key mapping. A key can act as speed-dial and supervisor (with light) to check if another IP-Phone is in call. A key can also send a DTMF sequence during call.

<u>CSV file transfers:</u>
For a large IP-Phone deployment, a serial auto-provision can be built from a CSV file.



- *Upload Serial provision CSV File*
  This feature allows to transfer CSV file from PC to S5000 and all the IP-Phone configurations may automatically be provisioned.
  The CSV file contents the following fields separated with "," or ";"
  1- Extension [numerical string]
  2- Phone type [Thomson-ST2030]
  3- Mac address [12 hexa digits]
  4- Display [text string without accentuated character]
  5- Number of lines [numerical value from 1 to 10]
  6- Time Zone [numerical value: see Time Zone table, empty for default]
  7- Language [numerical value: see language table, empty for default]
  8- Country tone [2 chars code: see country table, empty for default]
  9- DHCP [1=enabled, 0=disabled]
  10- Static IP address [empty if dhcp]
  11- Static IP mask [empty if dhcp]
  12- Static IP gateway [empty if dhcp]
  13- Forward type [None | Always | Busy | NoAnswer]

14- Forward address [destination number, empty if type=None]
15- Forward NoAnswer timer [numerical value in sec, min=10]
16- Ethernet Connection [numerical value: see Ethernet cctn table]
17- Interception group name

Depending of the type of phone, different Time Zone, language and Country Tone tables are used

Thomson and Panasonic Phones

Time Zone table

GMT-12:00	0	GMT+03:30	35
GMT-11:00	1	GMT+04:00	36
GMT-10:00	2	GMT+04:30	38
GMT-09:00	3	GMT+05:00	39
GMT-08:00	4	GMT+05:30	41
GMT-07:00	5	GMT+05:45	42
GMT-06:00	7	GMT+06:00	43
GMT-05:00	10	GMT+06:30	45
GMT-04:00	13	GMT+07:00	46
GMT-03:30	15	GMT+08:00	47
GMT-03:00	16	GMT+09:00	50
GMT-02:00	18	GMT+09:30	53
GMT-01:00	19	GMT+10:00	55
GMT	21	GMT+11:00	60
GMT+01:00	22	GMT+12:00	61
GMT+02:00	26	GMT+13:00	63
GMT+03:00	32		

Language table

English	0
Français	1
Espanol	2
Deutch	3
Italiano	4
Norsk	5
Russian	6
Portuges	7
Nedelands	8

Ethernet connection

Auto	0
100Mbps/Half	1
100Mbps/Full	2
10Mbps/Half	3
10Mbps/Full	4

Country Tone table

United States	US
France	FR
United Kingdom	GB
Deutchland	DE
Nederland	NL
Italy	IT
Spain	ES
Czech Republic	CZ
Portugal	PT
Slovenia	SI

Aastra Phones

In Aastra phones Time zone depends on the server NTP, To customize additional time zone parameters use Time one Table

Time Zone table

Def-12:00	0	Def +03:30	16
Def -11:00	1	Def +04:00	17
Def -10:00	2	Def +04:30	18
Def -09:00	3	Def +05:00	19
Def -08:00	4	Def +05:30	20
Def -07:00	5	Def +05:45	21
Def -06:00	6	Def +06:00	22
Def -05:00	7	Def +06:30	23
Def -04:00	8	Def +07:00	24
Def -03:30	9	Def +08:00	25
Def -03:00	10	Def +09:00	26
Def -02:00	11	Def +09:30	27
Def -01:00	12	Def +10:00	28
Def +01:00	13	Def +12:00	29
Def +02:00	14	Def+13:00	30
Def +03:00	15		

Country Tone table

US (English)	0
fr (Français)	1
es (Espagnol)	2
de (Deutch)	3
it (Italiano)	4
no (Norsk)	5
ru (Russian)	6
pt (Portuges)	7
nl (Nederland)	8

Language Table

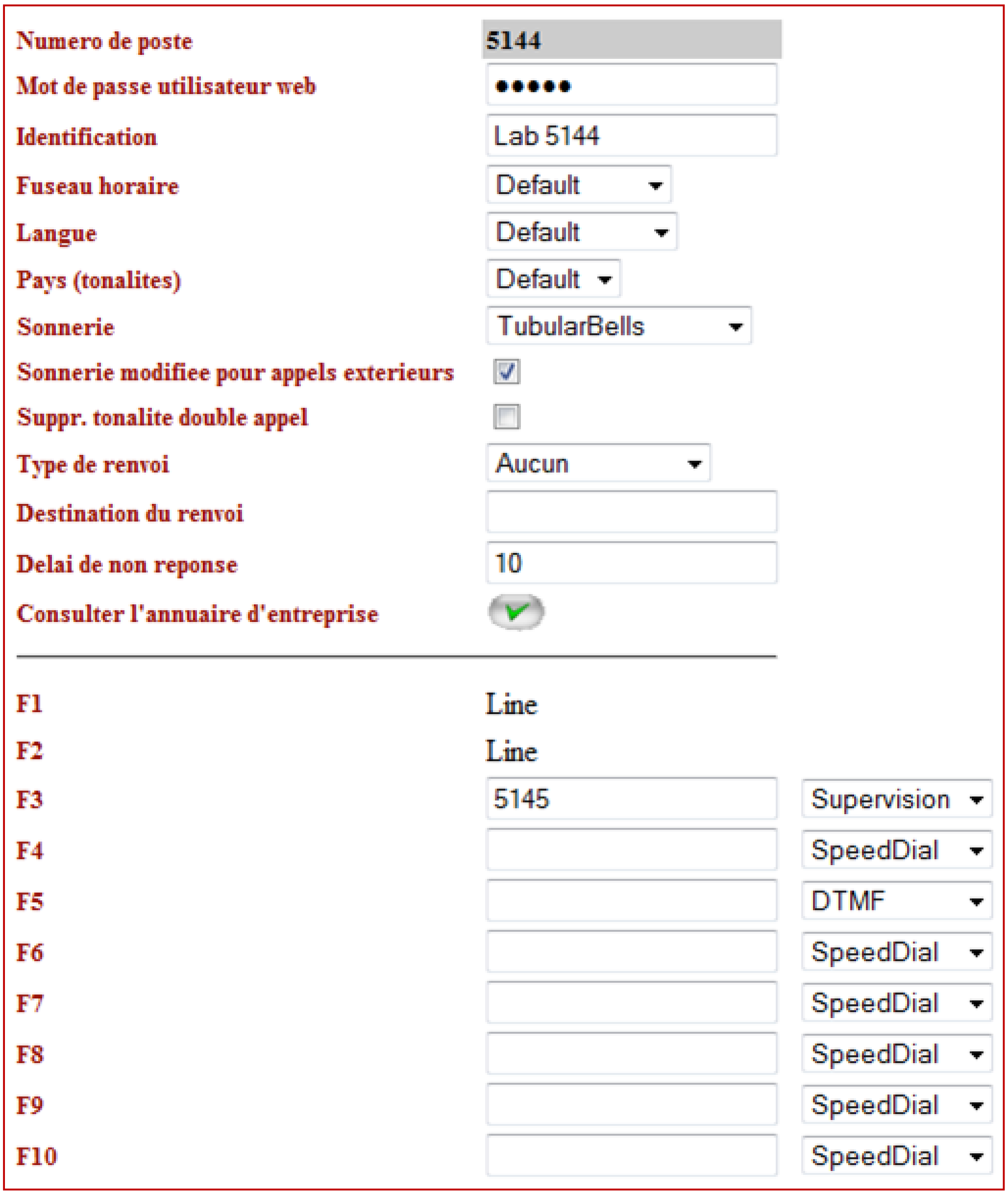
US	US
France	FR
UK	GB
Germany	DE
Nederland	NL
Italy	IT
Spain	ES
Czech Republic	CZ
Portugal	PT
Slovenia	SI

- *Save serial provision CSV File*

This feature allows to download all the auto-provisioned IP-Phones within a CSV file from S5000 to PC.
This file may be updated and uploaded later to the S5000.

5.4.4. User access to IP-Phone settings

When the phone auto-provisioning is enabled, a user can access to a restriction of the previous IP-Phone provisioning parameters to allow to personalize the ring melody, language, forwards, key, etc…



Numero de poste	5144	
Mot de passe utilisateur web	●●●●●	
Identification	Lab 5144	
Fuseau horaire	Default	
Langue	Default	
Pays (tonalites)	Default	
Sonnerie	TubularBells	
Sonnerie modifiee pour appels exterieurs	☑	
Suppr. tonalite double appel	☐	
Type de renvoi	Aucun	
Destination du renvoi		
Delai de non reponse	10	
Consulter l'annuaire d'entreprise	✔	
F1	Line	
F2	Line	
F3	5145	Supervision
F4		SpeedDial
F5		DTMF
F6		SpeedDial
F7		SpeedDial
F8		SpeedDial
F9		SpeedDial
F10		SpeedDial

Note: This page is displayed in French if selected language is French (or Default if default language is French).

5.4.5. Static Entities

Static entities allow sending calls to and/or receiving calls from non-registered endpoints (a gateway for example or another gatekeeper).
A remote site that has to be reached directly will be declared as a static entity **in H323 or SIP mode**.
A typical non registered endpoint will be declared as a static entity with a distinctive name and alias (same as name for example) and the endpoint IP address.
This endpoint will then be referred within the routing engine (embed Services) as its simple name.

There is 2 different static entity modes:

- OUT: Used to send outgoing calls to non-registered endpoint or to register to an operator registrar server.
- IN: Used to receive incoming calls from any or restricted non-registered endpoints. It acts as listener on a fixed port. It contents a list of remote authorize IP addresses.

Name	Mode	Sig.	IP address	Alias	IN Port	Options		
Addon (KO)	OUT	SIP	sipctrol3.add-on-line.net	0531117340				
Lab31VersEse (OK)	OUT	SIP	192.168.2.2	Lab31VersEse				
sip18000	IN	H323		lab30	18000			

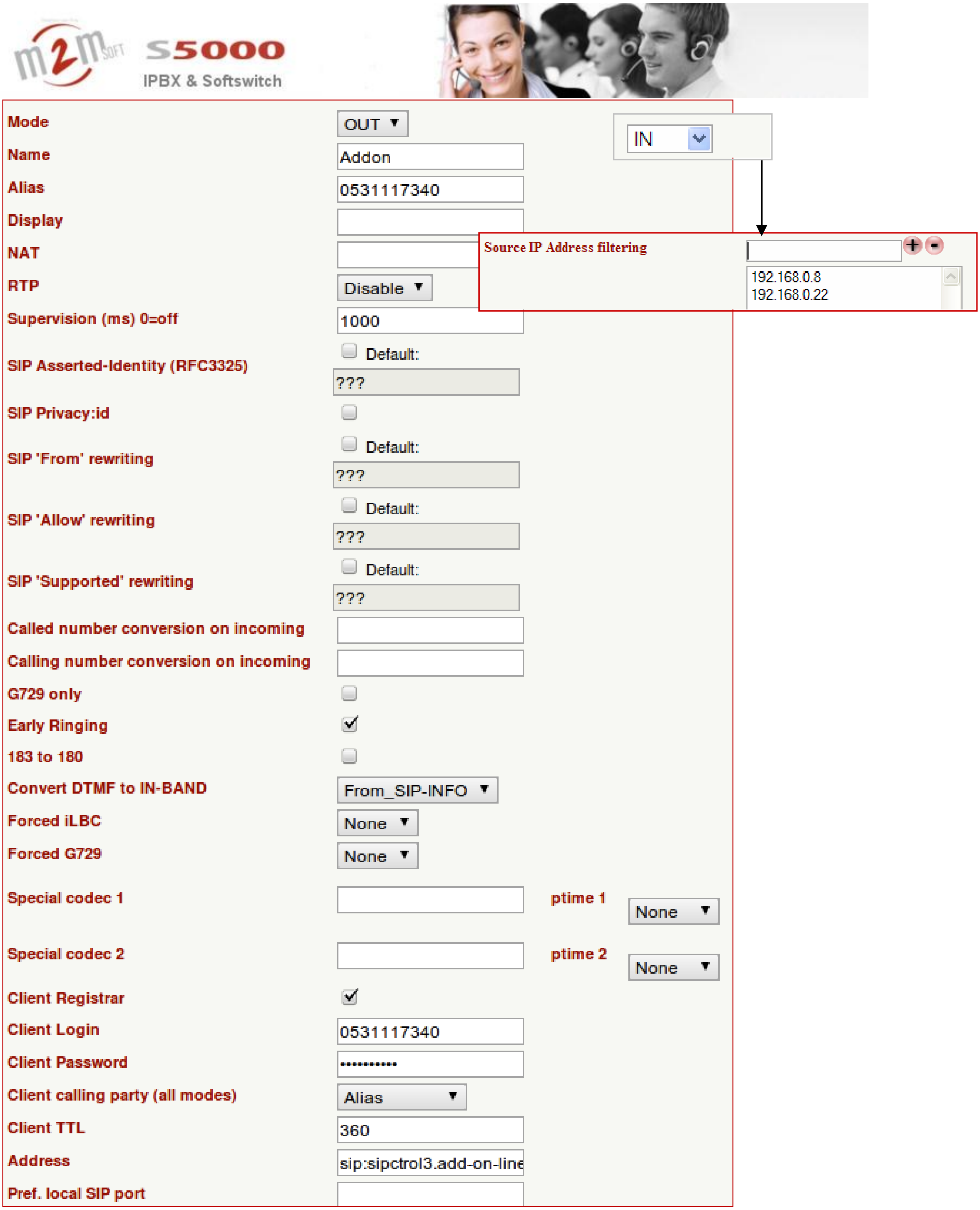


Fig.16 Static entity definition

- *Mode* OUT / IN. Adapts destination Address or Source addresses list.
- *Name* Static Entity name.
- *Alias* Can be the same as Name.
- *Display* Not yet implemented.
- *Q931 Port* Only used for IN mode.
  H323: define here the local TCP port to listen for incoming H323 direct calls. Common Q931 listener is 1720.
  SIP: define here the udp or tcp SIP listener.
  Forms are:
  sip:port to listen on specific UDP port.
  sip:tcp:port to listen on specific TCP port.

NOTE: use this to listen on non standard (non 5060) SIP ports.

- *RAS Port* Not used.
- *NAT* An IP address that is to be set within all outgoing packets to this StaticEntity address. The NAT parameter is generally set to the public IP address of your network on a StaticEntity that is associated to the WAN. All Q931 and H245 outgoing messages to your WAN have the gatekeeper address replaced by the NAT address. H245Routed must be enabled.
  SIP case: all IP addresses within SIP messages are replaced by the NAT field for SIP messages that go to the “Address” for StaticEntity OUT or for all SIP messages that go to the party that initiates an incoming call that enters here as a StaticEntity IN.
- *RTP* When enabled, this force all incoming media (RTP-UDP) packets from the remote connection to be proxyied within the S5000 and thus directed directly to the right internal endpoint. Two channels, one audio and one video are translated per call.
- *Supervision* (Only for mode OUT) To enable a polling to check if the endpoint is alive.
  This feature is used by Routes/Trunks to select a valid destination of call.
- *SIP Asserted-Identity*
- *SIP Privacy:id*
- *SIP ’From’ rewriting*
- *SIP ‘Allow’ rewriting*
- *SIP ‘Supported’ rew.*
- *Called num. conv.*
- *Calling num. conv.*
- *G729 only*
- *Early Ringing*
- *183 to 180*
- *Convert DTMF to IN-BAND*
- *Forced iLBC*
- *Forced G729*
- *Special codec 1&2*
- *Ptime 1&2*
- *Client Registrar* Enable/Disable registration to remote SIP provider (digest mode).
- *Client Login* Login of the account as given by your remote SIP provider.
- *Client Password* Password of the account as given by your remote SIP provider.
- *Client calling party* *Source alias sent toward provider.* Alias or Anonymous or Transparent (Transparent=initial caller alias maybe processed by embedded service).
- *Client TTL* TTL for registration to remote provider.

- *Address* (Only for mode OUT) Destination endpoint IP address.
  For SIP endpoint
  Address must have form "`sip: address`".
  NEW: NAPTR search. The form `sip:enum:<base suffix>` where <base suffix> is the E164 database to search thru DNS (can be e164.arpa for example). This enable a DNS search for this phone number. See "Advanced routing: DNS chapter.
  For H323 endpoint:
  Address is in the form "address".
- *Pref. local SIP port* Port can be added (:port). The default one is 5060 for SIP, UDP. and port is 1720 for H.323, TCP.

RSVP parameters (RFC2205)

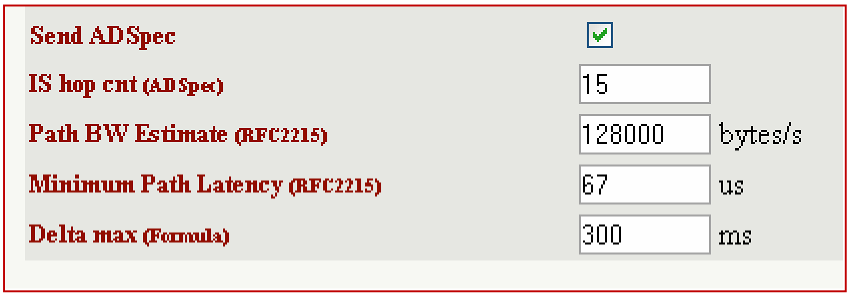
Only available with RSVP option. See § 6.9 for more information about RSVP feature.

RSVP	Enable
Time values	30000 ms
Style	SE
Send FlowSpec	☑
Token Bucket rate (Flowspec/TSPEC)	200 bytes/s
Token Bucket size (Flowspec/TSPEC)	1000 bytes
Peak data rate (Flowspec/TSPEC)	2000 bytes/s
Guaranteed rate (RSPEC)	200 bytes/s

- *RSVP* Enable/Disable RSVP feature.
- *Time values* Refresh period (in ms) for rsvp reservation handled from here. S5000 automatically refreshes the reservation with new Path messages.
- *Style* RSVP resources reservations can be distinct or shared and the Style defines this. Values are: FF (Fixed Filter), WF (Wildcard Filter) and SE (Shared Explicit). FF is the most recommended option as it reserves a distinct path for each RTP flow.
- *Send FlowSpec* Enable/Disable FlowSpec (See RFC2205 and IntServ information).
- *Token Bucket rate* Bytes/sec (See RFC2205 and IntServ information).
- *Token Bucket size* Bytes (See RFC2205 and IntServ information).
- *Peak data rate* Bytes/sec (See RFC2205 and IntServ information).
- *Guaranteed rate* Bytes/sec (See RFC2205 and IntServ information).

<u>**RSVP advanced parameters for OPWA (RFC2210 and RFC2215)**</u>

We implement a two-passes RSVP as defined within RFC2210 and uses the ADSpec parameter informations.



- *Send ADSpec* Enable/Disable OPWA feature by sending/not sending ADSpec information element within Path messages. The following parameters define the initial ADSpec content.

- *IS Hop Cnt* (see RFC2215) IS Hop counter. The number of 'integrated services' (IS Hops) aware nodes is updated here along the Path.

- *Path Bandwidth estimate* Initially required Bandwidth advertised within Path message

- *MinimumPath Latency* The initial smallest delay added to process the packet at that very node. Other crossed nodes will update this.

- *Delta max* End to end maximum required VoIP voice delay. This will be used to compute (magic formula here !) the new FlowSpec parameters within Resv message

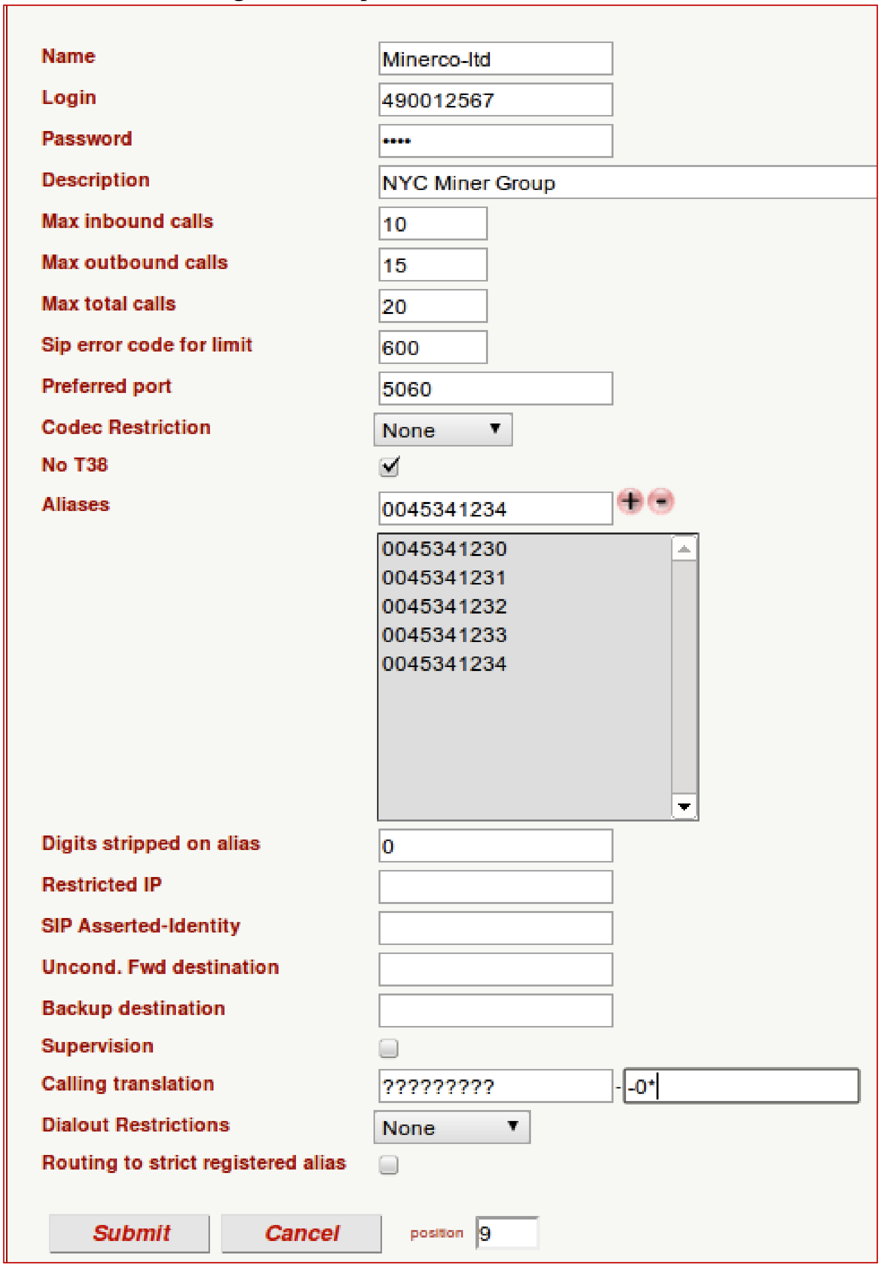
5.4.6. SIP Accounts

The S5000 manages secured SIP registrations with Digest mode.
SIP Account let you define accounts where you give a login and a password to your users to register and place calls.
Several SDAs can be defined per SIP-account allowing an incoming call to be routed to a sip account (to an IP destination) if the destination number matches any of the defined alias for the account and if the account is registered at that time.

This is an Operator feature.

Name	Login	Calls out	Calls in	
M5000	M5000	0/10	0/-1	
Minerco-ltd	490012567	0/15	0/10	
A6000	A6000	0/200	0/-1	

. Fig. Global Sip Accounts view



- *Name* Name this entry.
- *Login* Login information of the account. Can be a phone number. The endpoint will need to enter this to register here.
- *Password* Account password information, goes with the login.
- *Description* A free entry to note informations about this account (company name for example)

- *Max inbound calls* Maximum concurrent incoming and established calls
- *Max outbound calls* Maximum concurrent outgoing and established calls.
- *Max total calls* Maximum concurrent both way calls.

- *SIP err. Code*
- *Preferred port* Use (if not -1) this port to route calls to remote party (call will be routed on the registered endpoint remote address and this port). Usually one have Nated this port on the customer router.

- *Codec Restriction* To only accept call from user which contain the specified codec and remove the others.

- *No T38* If enabled, do not allow calls with T38 codec to establish.

- *Aliases* The list of SDA managed by this account. An incoming call for one of those will be routed (sent) to this account. (See EmbeddedService SIPACCOUNT)
- *Restricted IP* To only accept the specified IP of account.

- *SIP Asserted*

- *Uncond Forward dest.* When set, direct all calls to this account, any time, to the specified number.

- *Backup destination* In case the S5000 could not route calls for these SDA to the destination, here is an IP address to send the call as an alternate route.

- *Supervision* To enable a polling with endpoints and forward calls to backup destination if polling is lost.

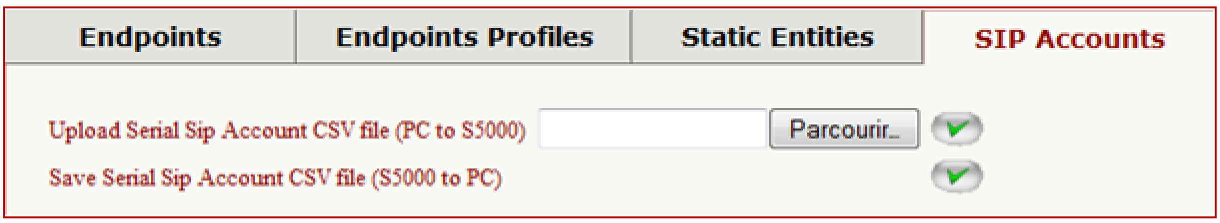
- *Calling translation* To translate source alias of incoming call.

- *Dialout Restrictions* To apply a call restriction defined at § 5.6.6.

- *Routing to strict Alias*

- *Position* Position in the SIP Account list.

CSV file transfers:
The SIP Accounts can be imported and exported from/to a CSV file.
To get the field contents export first.



5.5. *Calls page*

Calls page is subdivided in several sections selectable with tabs.

Active Calls	Statistics	Current CDRs	Archived CDRs

This page allows supervising and releasing active calls, control joined calls trough media entities, view current CDRs (Call Detail Records), download and suppress archived CDR.

The following sub-chapters describe all the features.

5.5.1. Active Calls

State	Calling	Called	CallId	Extra	Current duration	
[Connected]	52	5179@192.168.3.34	02B22F75DB00001033A55634343434EF MEDIA02B22F75DB00001033A55634343434EF		Tue Jun 18 2013 13:30:28.372(0:0.23)	disconnect
[CONNECTED]	5181@192.168.3.34	5180@192.168.3.34	1173f259-c0a80101-0-b7@192.168.3.42 K173500485910011	MTP	Tue Jun 18 2013 13:28:23.366(0:2.22)	disconnect

This sample shows 3 call types:

- The first one is a pure H323 ⬄ H323 call,
- The second one is a pure SIP ⬄ SIP call (with original + final call-Id),
- The last one is a mixed SIP ⬄ H323 call (with original + final call-Id).

Note: In IPBX mode, a connected call is a junction of 2 half calls connected to 2 media entities. However in this case only one line is displayed in the main view with both real endpoints. But it is possible to have the half calls display (with media entities) by clicking on "Call-legs view".

mtpName=mtp0,
channels=0:1

The STATE column displays the call status: [Establishing], [Ringing], [Connected].
The CALLING column displays the calling party number (or source alias).
The CALLED column displays the called party number (or destination alias).
The CALLID column displays the call reference string.
The EXTRA column displays optional information such as MTP in case of Media Termination Point used for the call (see MTP § 5.7.2). When mouse pointer is over MTP field an info box is displayed:

The pause command allows to split call participants and connect each of them to a media entity.
A join command join will be displayed to re-join both participants together.
These commands are supported for pure H323 calls only.

The disconnect command release the call.

5.5.2. Daily CDRs

Date ▲	Duration	Calling	Called	Initial called	Status	CallId
Tue Jun 18 2013 13:38:30.788	0:0.15	5181	911	911	MC	156f915e-c0a80101-0-cb@192.168.3.42
Tue Jun 18 2013 13:33:33.396	0:0.15	5181	5180	5180	SJ	1399cda2-c0a80101-0-c2@192.168.3.42
Tue Jun 18 2013 13:33:05.985	0:0.19	5181	5180	5180	SJ	1333f983-c0a80101-0-c0@192.168.3.42
Tue Jun 18 2013 13:30:28.372	0:0:41	52	5179	5179	MJ	02B22F75DB00001033A55634343434EF
Tue Jun 18 2013 13:29:34.383	0:0.1	5182	6000	6000	SJ	123e8f26-c0a80101-0-c0@192.168.3.48
Tue Jun 18 2013 13:29:11.298	0	5182	5183	5183	SE	11dc6d99-c0a80101-0-be@192.168.3.48

This area displays the list of Call Detail Records of the current day. By default the list is sorted from newest to oldest. The arrows ▲▼ allows to change the order.

The arrows allows to browse the page within the same day.
A CDR is created for each call terminated. The CDRs are stored in CSV format within a file archived every day at 12:00pm.
The table extracts the main fields of the CDRs (see § 5.5.3 for the full CDR definition):

- Date and Time from the starting of the call.
- Call duration.
- Calling Party Number (From Alias).
- Called Party Number (To Alias) which accepts the call.
- Initial Called Number.
- Status (1)
- Call-ID (The original call part reference in case of iPBX mode).

(1) The status is displayed with 2 chars: The 1st one is the protocol; the 2nd one is the status at the end of the call:
→ Protocol: S=Sip, H=323, M=Mixed
→ Status: E=Establishing, C=Connected, J=Joined
(Ex: HE=H.323 establishing, SJ=Sip Joined …).

5.5.3. Archived CDRs

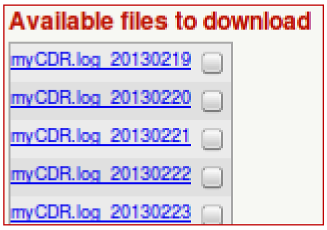


Fig.17 Archived CDRS pages

Cyclic file writing:
According to disc quota and number of local files allowed for the CDR (see General Parameters § 5.3.1), the CDR will be logged in a cyclic way according to the disc quota allocated.
A daily swap is also performed and the CDR counters are reset at midnight.
A new CDR files family is built every day.
An administration task is to define how many CDR days you want to keep within your server. All of the available files are downloadable through the S5000 web interface.

The daily CDR file has the following name: CdrFileName_YYYYMMDD where:
- CdrFileName is configured at General Parameters: [CDR File Name] parameter.
- YYYY is the current year
- MM is the current month
- DD is the current day

The CDR file name can have an optional suffix such as _0 or _1 according the [CDR File Size] and [CDR File Number].
From this area it is possible to download CDR files and to delete them.
CDR files are in CSV (Comma Separated Values) which can be ridden by standards spreadsheets.

Fields definition:
- dateTimeOrigination: The date and time when the call request started.
- OriginalCallId: Main call reference or 1st call part reference in case of iPBX mode.
- OrigIpAddress: Caller IP address.
- OrigIpPort: Called Call signal port.
- CallingNber: Caller E164 number.
- DestIpAddress: Destination IP address.
- DestIpPort: Destination Call signal TCP port.
- OriginalCalledNber: The original called number as dialled in the call request.
- FinalCalledNber: The real called number as replaced (if any) in case of redirect or transfer.
- DateTimeConnect: The date and time when the call connect occurred.
- DateTimeDisconnect: The date and time when the call release occurred.
- Duration: The duration (hours/minutes/seconds) of the call when established.
- TimeToRingDuration: The duration (seconds/ms) to get the Alerting from the Setup (H323 only).
- ReleaseCause: The release cause as advertised in Q931 or SIP Cause element.
- Status: XY. X:S=SIP, H=H323, M=Mixed, Y:E=Estab, C=Connect, J=Joined.
- FinalCallId: The 2nd call part reference in case of iPBX mode.
- Source Sip Account: The name of caller SIP account if exists
- Dest Sip Account: The name of called SIP account if exists

5.6. *Embedded Services page*

Embedded Services page is subdivided in several sections selectable with tabs.

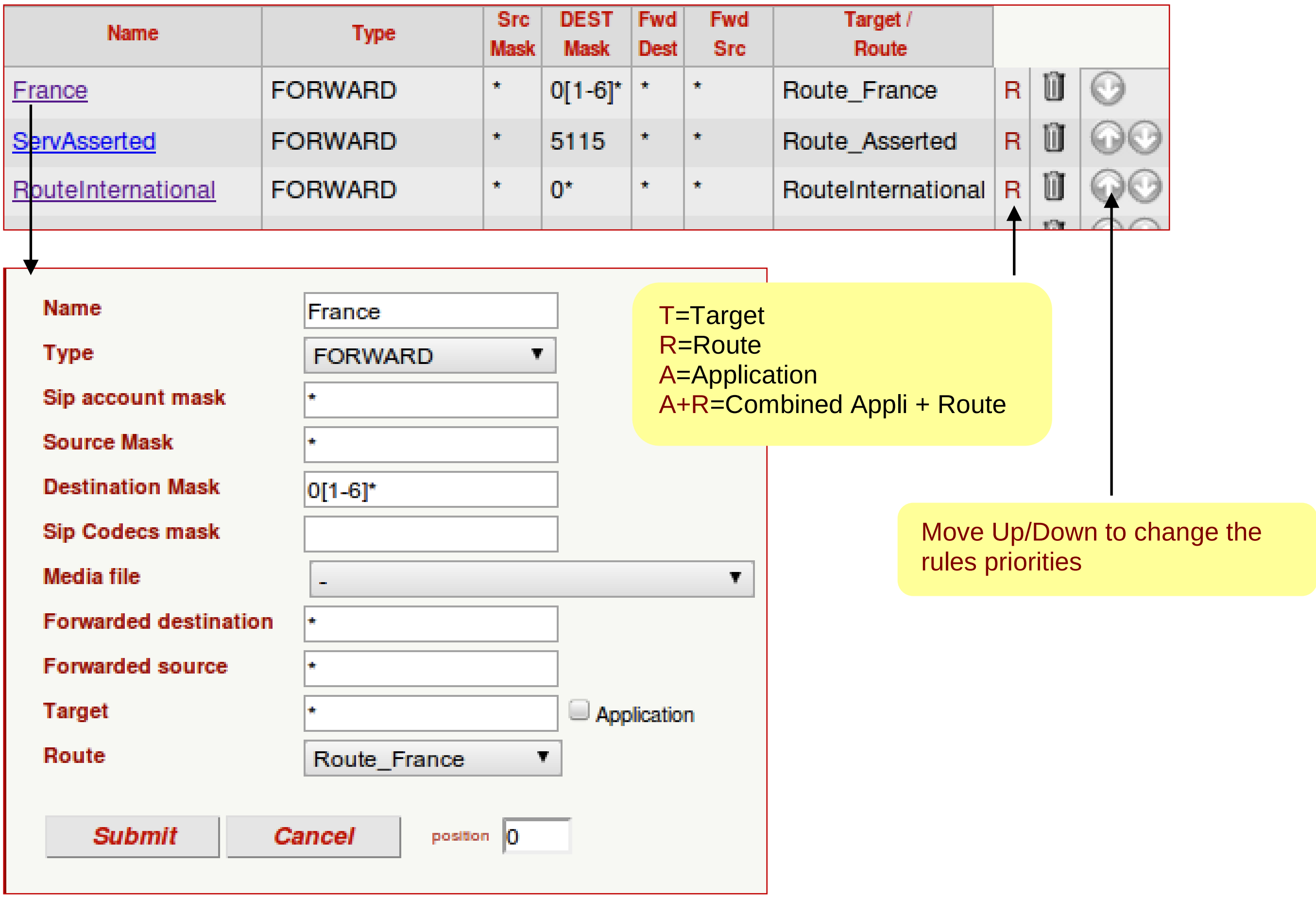
Services	Routes	Trunks	TrunkMap	Restrictions	SpeedDials

This page allows configuring registration controls, digits transformation, simple and advanced routing, call restrictions and Speed-dials.

Call as well as SMS Short messages routings can be handled precisely through the definition of Embedded Services.
The following sub-chapters describe all the features.

5.6.1. Services

This is one of the most used features: this allows you to define precise routing rules based on called numbers and calling numbers.
Define a name for your rule, a called number pattern (destination Mask), a calling number pattern (source mask) and some actions to be made:



Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route			
France	FORWARD	*	0[1-6]*	*	*	Route_France	R		
ServAsserted	FORWARD	*	5115	*	*	Route_Asserted	R		
RouteInternational	FORWARD	*	0*	*	*	RouteInternational	R		

Fig.18 Embedded services page

- *Name* Service name.

- *Type* FORWARD: the call is always forwarded (to Target/Route/Application).
ALTERNATE: the call is forwarded to a Target only when the destination is not currently registered.
RREJECT: the endpoint is not allowed to register the system.
RACCEPT: the endpoint is allowed to register the system. (H323 mode).
SIPACCOUNT: routing **to** a sip account rule. Consider the destination mask to match a sip account name (as set in creating sip account within the S5000). Any SDA defined within a matching sip account name will be routed as destination.
SIPACCOUNTSRC: routing **from** a sip account rule. Consider the Source mask to match a sip account name (as set in creating sip account within the S5000). If the caller is identified in a sip account whose name matches the source mask, then it is looked at the destination party.
This later can be used by operator to have different routing policies according to sip account families. (The A_* sip accounts and the B_* sip accounts for example)

- *Sip account Mask* For type=SIPACCOUNTSRC: the regular expression here matches a sip account name of the caller.

- *Source Mask* For type=FORWARD or ALTERNATE or SIPACCOUNT: the regular expression that is
expected to match on the calling number.
Default is '*': all calling numbers, no restriction
Example: 22* means that all IP phones with number starts with 22 will match.
It can be combined with Destination Mask allowing for very complex call filtering based on terminal selection and called numbers.

- *Destination Mask* For type=FORWARD or ALTERNATE or SIPACCOUNTSRC: the regular expression that is expected to match on the called number.
For type=RREJECT: the regular expression that is expected to match the source alias.
Example: 78*: will reject registration of 78, 781…endpoints.
See *Forward Pattern* to reject vendor specific endpoints.
For type=SIPACCOUNT: the regular expression used to match a sip account name and all of its SDA.

- *Codecs Mask* SIP only.
Specify a list of codecs expected for the routing to match.
Codecs mask can be any combination of:
+G729 expect G729 codecs within SDP
+G711A expect PCMA within SDP
+G711U expect PCMU within SDP
+G7231 expect G723 within SDP
Leave field blank to avoid codec control.
Combine multiple expectations as +G7231+G729 that defines that the call is expected to advertise G723 AND G729 codecs in order to match the embeddedService rule.

- *Forward Pattern* For type=FORWARD or ALTERNATE. Modify the called number.
. (DOT) keep the digit from the original called number.
+ advance 1 digit, skip the corresponding digit in the original numbers.
[**0-9**] digit which replaces the corresponding digit position.
***** to keep all digits without any change.

- *New Calling Number* For type=FORWARD or ALTERNATE or SIPACCOUNTSRC : modify the calling number.
  . (DOT) keep the digit from the original called number.
  + advance 1 digit, skip the corresponding digit in the original numbers
  [**0-9**] digit which replaces the corresponding digit position.
  ***** to keep all digits without any change.
  When *type*=RREJECT and *Destination Mask*='vendor', *New Calling Number* must contain the productId (see Enpt. Registr. Ctrl § 6.6).
  (Example: *New Calling Number*=Microsoft NetMeeting' will reject all Microsoft endpoints).

- *Target* To forward call to a registered endpoint, Target=<@alias> (ex: @5182).
  To forward call to a non-registered endpoint (such as gateway or remote S5000) create a static entity and set Target=<StaticEntity name>.
  To forward call to an application set Target=<Application name> and check "***Application***" check box.
  To forward call to a Route, let Target=* and select the Route (see below).
  If the destination is not to be forced (embeddedService used for change calling or called number) let Target=* .
- *Route* This selects an advanced routing instead a simple destination used by a *Target*. Create Trunk(s) and Route and select the Route displayed in the list box (see § 5.6.2).
  It is possible to combine Application and Route. In this case the call is first forwarded to the application and then forwarded to the Route process.

5.6.2. Routes

The Routes allows to forward call according advanced routing rules, such as combined load balanced and backup trunks managing Busy destinations, No-Answer, call limitations.
See Advanced Routing chapter § 6.7

Name	Mode	Trunk 1	Trunk 2	Trunk 3	More	
Route_France	BKP	FranceTelecom	GSS			
RouteInternational	BKP	MXTelecom	GSS			
Route_Ese	BKP	vers_Esesip				

Fig.19 Routes page

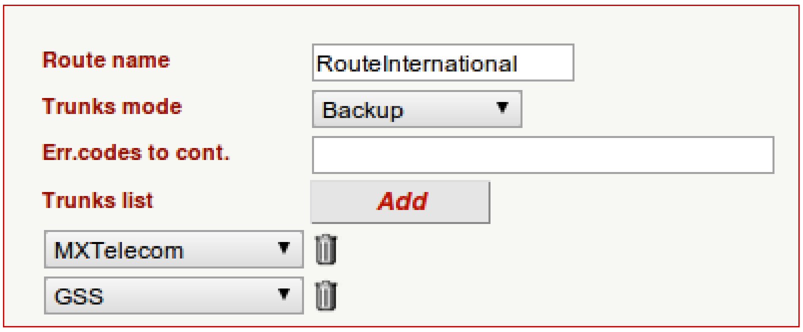


Fig.20 Route details page

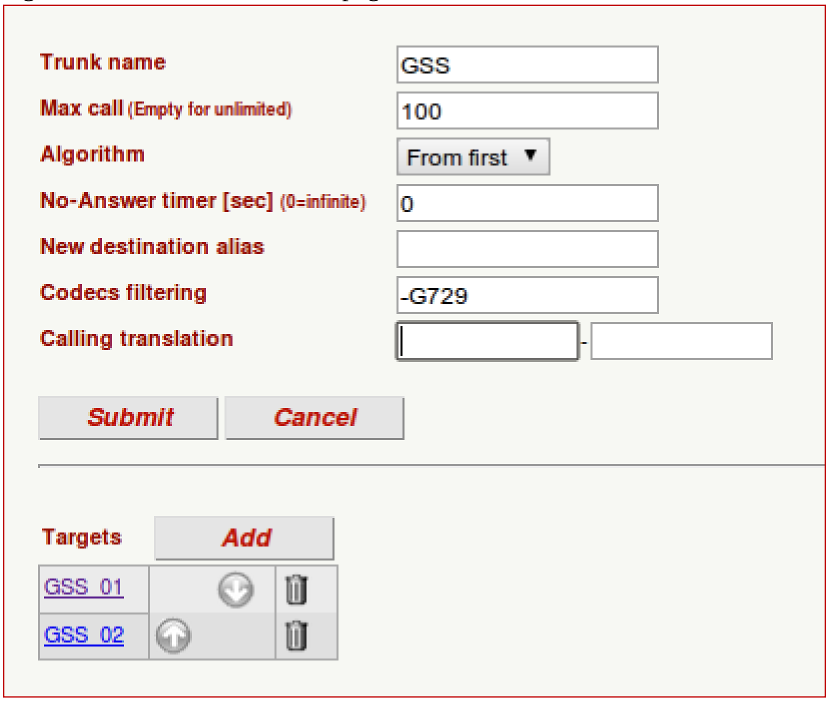
A route contents ordered trunks. The route uses the first trunk. When this trunk is unavailable or busy the route tries to use the next trunk. A trunk (see below) contents one or several targets (SIP or H323 destinations).
The list boxes allow selecting existing trunks, so the trunks must be created before.
To add an alternate trunk click '***New***' button, to remove a trunk from the route click 'remove' link.

5.6.3. Trunks

The Trunks are selected by Routes and contain one or several targets as VoIP SIP or H323 destination. The targets are load balanced destinations and are scanned according the Algorithm parameter. The call capacity can be limited in the trunk to allow the use of an alternate trunk in the route. Another target within the trunk is tried when a call is released without connection (busy cases, or unreachable destination…). Another target within the trunk can be tried when the call is not answered after a delay (optional), it is useful for Contact Centers.
See Advanced Routing chapter § 6.7

Name	Max call	ALGO	Target 1	Target 2	Target 3	More	
FranceTelecom	Unlimited	Rotary	@4000				
Trunk_Asserted	Unlimited	Ffirst	@5100				
GSS	100	Ffirst	GSS_01	GSS_02			

Fig.21 Trunks and trunk details page



- *Trunk name* Trunk name used by Routes
- *Max call* To limit the call capacity within the trunk (for any target). Let the field empty for no limitation.
- *Algorithm* ***FromFirst***: The targets are scanned sequentially always beginning from the first target in the list.
  Rotary: The targets are scanned sequentially in load balancing mode.
  MultiRing: The call is forwarded to all targets at the same time.
- *NoAnswer timer* Try the next target when no answer after the timer (sec). 0=infinite.
- *New destination alias* To change the final destination within the trunk. If this field contains "*" the string will be added as prefix to the initial dialled number.

- *Codecs filtering* SIP calls only. Set here any codec names you may want to suppress for the outgoing call. (to force connection to establish on a specific codec)
  Leave the field blank to keep the original codecs.
  -G729 suppress the G729 codec from the SDP (named G729)
  -G7231 suppress the G723.1 codec (named G723)
  -G711A suppress the G711A law codec (named PCMA)
  -G711U suppress the G711Ulaw codec (named PCMU)
  Combine any of these to suppress multiple codecs. (-G711A-G711U ...)

Add and remove targets to/from trunk, and change the order with arrows .

Edit targets list:

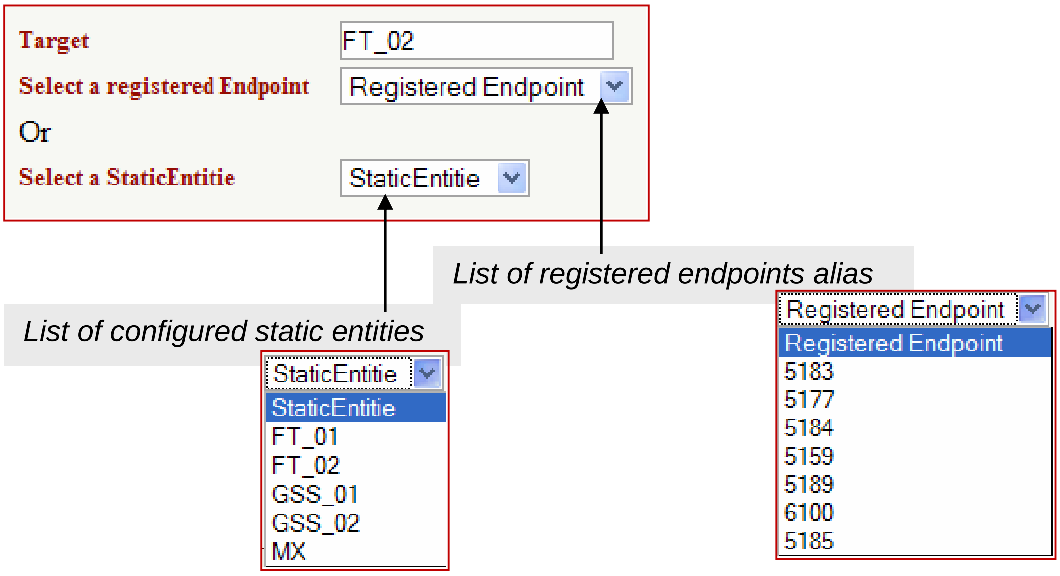


Fig.22 Trunk's targets list details page

Select a target and click ***Submit*** button.

5.6.4. Trunks statistics

This area displays some counters about calls in each configured trunks.

Refresh | **Reset counters** | **Back to trunks list**

Trunk	Nb of targets	Max capacity	Active Calls	Max call	Total of calls
Trunk_OUT	1	Unlimited	0	0	0
Trunk_MSG	1	Unlimited	0	0	0
Trunk_A6000	1	Unlimited	0	44	252

Fig.22 Trunk statistics page

- *TRUNK* Trunks name.
- *NB OF TARGETS* Number of targets within the trunk.
- *MAX CAPACITY* Max capacity of calls configured.
- *ACTIVE CALLS* Number of established calls in the trunk.
- *MAX OF CALLS* Maximum simultaneous established calls in the trunk.
- *TOTAL OF CALLS* Counter of calls since the last reset.

Click ***Refresh*** button to update counters, and ***Rest counters*** button to reset 3 last columns values.

5.6.5. IP/Trunks mapping

This table allow to link a remote IP address with a trunk to manage incoming calls limitation.

IP/Trunks Mapping for incoming call limitation

Add

Source IP	Trunk
192.168.3.32	FranceTelecom
192.168.0.38	MXTelecom

Source IP	192.168.0.31
Trunk	Trunk_PABXAERO

5.6.6. Restrictions

This area allows to define restrictions for dialout calls. This restriction can be applied to EndPointProfiles to restrict calls from SIP or H323 endpoints (see § 5.4.2).

Name	Mode	Patterns
Restric_ULNPI	Accept	01[78],011[25],51,06*
A518*	Accept	
Reject0605	Reject	

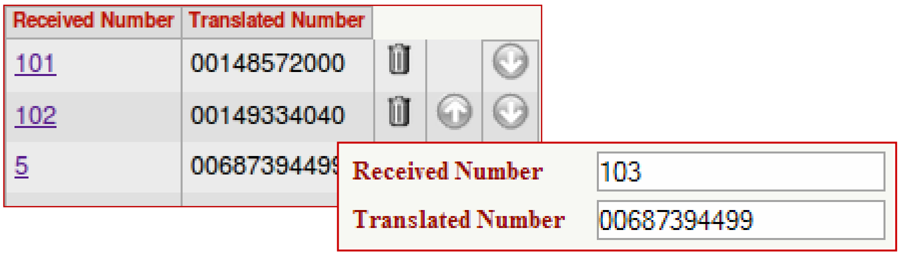
Restriction name	Restric_ULNPI
Mode	Accept
Patterns	01[78],011[25],51*,0*
Forward (empty if simple reject)	911

- *Restriction name* Name used by EndPointProfile definition.
- *Mode* ***Accept***: To only accept calls matching patterns and reject others
  Reject: To only reject calls matching patterns and accept others.
- *Patterns* Regular expressions matching destination field of the call. The patterns are separated by coma.
- *Forward* Optional. Destination address to forward the call in case of reject (can be a local MediaEntity). If this field is empty the call is simply released.

5.6.7. SpeedDials

Manual configuration:

This area allows to define a list of global speed-Dials. This list can be uploaded from a CSV file (see CSV file transfers later in this chapter).

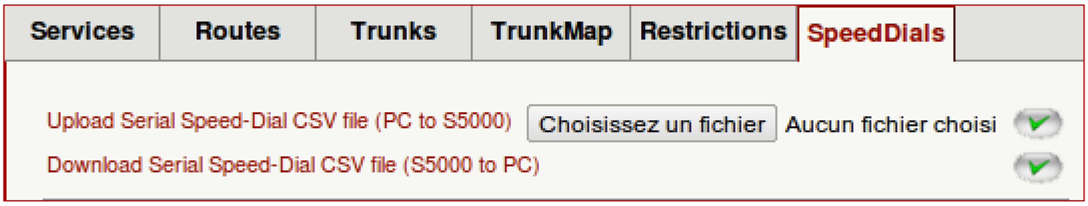


- *Received Number* Dialled short number
- *Translated Number* Forwarded destination number.

Note: The translation is processed before Embedded Services.

CSV file transfers:

It is possible to transfer either (upload from PC to S5000 and download from S5000 to PC) the complete list of speed-dials.



- *Upload Serial Speed-Dial CSV File*

  This allows to upload into S5000 a list of global speed-dial from a CSV file build on PC.
  The CSV file contents the following fields separated with "," or ";"
  1 Received dialled (short) number
  2- Forwarded translated number

- *Download Serial Speed-Dial CSV File*

  This allows to download from S5000 the list of speed-dial into CSV file on PC.
  This file can be updated and uploaded later to S5000.

5.7. *Media page*

Media page is subdivided in several sections selectable with tabs.

This page allows configuring Media entities (embedded media file server), MTP (RTP relay) and MTP rules. Also it allows to record audio files.

The following sub-chapters describe all the features.

5.7.1. Media Entities

Media Entities are internal audio resources (working as internal endpoint) used to connect calls with play files features. A media Entity is defined with a name/alias and a "play file" to be heard whenever someone connects (dial in or dial out) to this alias number. (G711A, G723.1 and G729 law format play files). Media Entity works with H323 and SIP protocols at the same time.

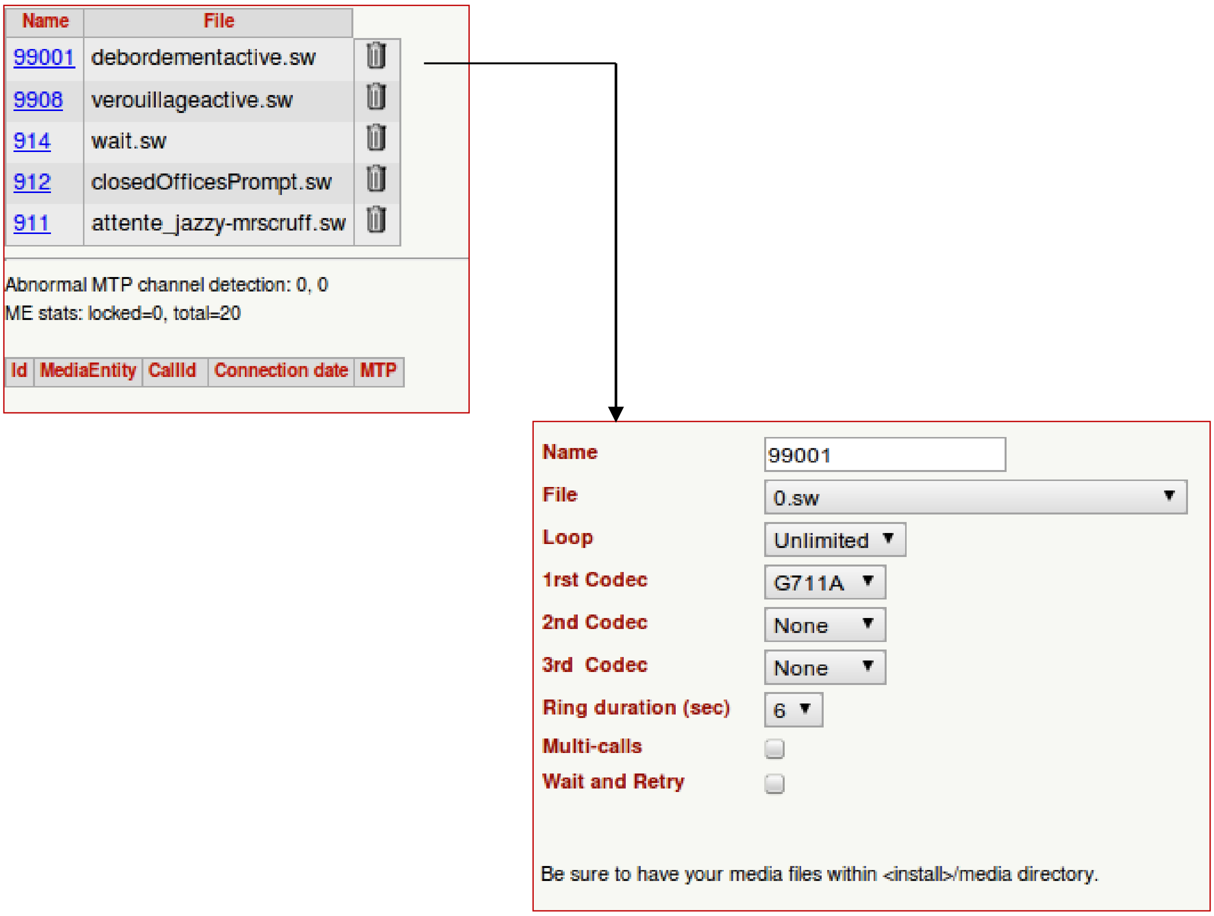


Fig.23 Media entity detail page

- *Name* Media entity extension.
- *File name* File selection. It must be within <install>/media directory. Only G711 filesare listed (G729 and G723 files are automatically selected according call capabilities and codecs configuration (see below).

G711 file suffix is .sw
G723 file suffix is .sw.g723
G729 file suffix is .sw.g729

- *Loop* ***Unlimited*** repeats, or ***1 until 4*** repeats.
- *Codecs* Select preferred codecs order among G711A, G723.1, and G729.
- *Multi-calls* *Allow concurrent calls for the same media entity.*
- *Wait and Retry*

5.7.2. Media Termination Points

MTP allows defining RTP relays to translate media flows through fixed locations. This is useful for operators who want to mask their customers IP addresses. It is also used for enterprises extended services such as Hold, Transfers, call parks, etc….
We can define local MTP (in the same host than S5000) and remote MTP (to prevent RTP flows over low rate WANs). MTP Rules define in which cases MTP are to be used, in a call per call basis.
See MTP chapter for more details § 6.3.
Optionally:

- *MTP can handle SecureRTP vs RTP conversions. (RFC 3711)*
- *MTP can handle inband DTMF conversion from out of band RFC2833 or SIP INFO*

Name	IP address	TCP Port	1ST RTP	State		Current	Version	
mtp0	192.168.3.31	29000	29000			0/100	1.126	
Paris	192.168.3.30	30000	30000			0/0	?	

Fig.24 MTPs status view

Status:

: MTP connected.

:MTP not connected

: Local MTP stopped.

: Start a local MTP

:Stop a local MTP

Name	mtp0		
IP address (* for local)	*	All IP binding	
TCP Port	29000		
First RTP Port	29000		
NAT allowed	Yes		
Kill S5000 if MTP lost	No		
Dejitter buffer IN (ms)	Auto		
Dejitter buffer (ms)	100		
Remote RTP port discovery			
AMP buffer mode			
Max channels (only for local MTP)	100		

Fig.25 MTP detailed view

- *Name* MTP name used by MTP Rules.
- *IP address* MTP host address (or * if this MTP is in the same host as S5000).
- *All IP Binding* Checked to
- *TCP Port* TCP supervising channel.
- *First RTP Port* UDP/RTP port of the first media channel.
- *NAT allowed* To prevent automatic NAT (see figure below).
- *Kill S5000 if MTP lost* Kill S5000 if MTP cannot be connected. Only to be used with the M2M-ControlCenter which surveys and restarts the applications.
- *Dejitter buffer IN*
- *Dejitter buffer* RTP dejitter buffer size (in ms) from 0 to 1000ms.
- *Remote RTP port discovery* Checked to auto detect the remote RTP port.
- *AMP buffer Mode* Checked to
- *Max channels* Set the number of local MTP channels. Not used for remote MTP.

NAT for MTP

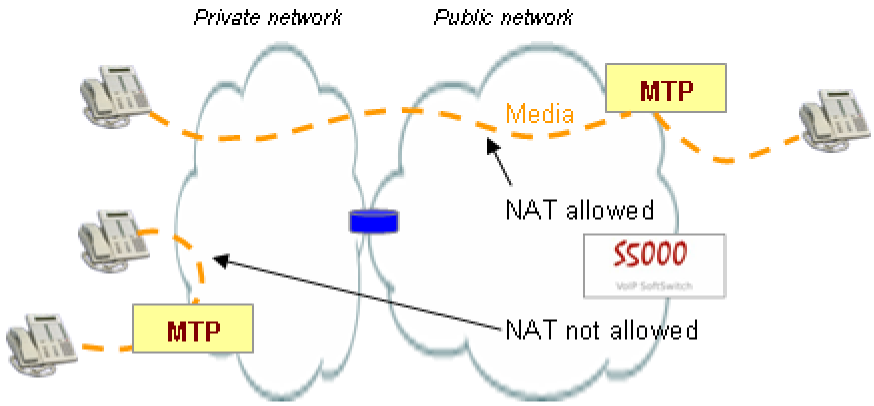


Fig.26 MTP for intelligent natting

5.7.3. MTP Rules

MTP Rules define in which cases MTP are used and how media channels are selected. When a call matches a source AND a destination condition, a corresponding MTP is selected with channels allocation policy.

Name	Source	Dest	MTP	CH. IN	CH. OUT	
IntraParis	40*	40*	Paris	Auto	Increase from 0	
Default	*	*	NO_MTP	Auto	Increase from 0	

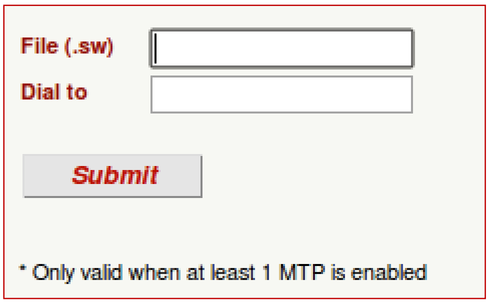
Rule name	Default	
Source pattern	*	
Destination pattern	*	
MTP	Local	
Channel IN (-1=auto)	-1	(RTP:28000)
Channel OUT	Increase from 0	(RTP:28000)
Jitter Buffer rule	normal	

Fig.27 MTP rules page

- *Source pattern*. Regular expressions matching either:
  - Source E164 alias
  - Source sipAccount with form “SIPACCOUNT_<account name>”
  - Remote RTP audio port with form “RTP_<port>”
- *Destination pattern*. Regular expressions matching either:
  - Destination E164 alias
  - Destination sipAccount with form “SIPACCOUNT_<account name>”
  - Target staticEntity with form “SE_<staticEntity name>”
- *MTP* MTP selection
  (“NO_MTP” allows to create a rule which prevent MTP use).
- *Channel IN* Forced channel number of incoming call-leg (RTP port=first RTP + 2*ch).
- *Channel OUT* Channel allocation way for outgoing call-leg.

5.7.4. Recorder

Recorder allows to record announce files in G711Alaw format. S5000 places a call toward your phone and a prompt invites you to record announce. At the end of announce you press '#" key, then announce is played and the call released. The file is saved within media directory and it can be listed in all media lists (MediaEntities, IPBX…).



- *File name*. File name with .sw extension (extension added if omitted).
- *Dial to* Alias of phone which receive call and deposit announce.

IMPORTANT: This feature can be used only if an MTP is enabled (the MTP Rules are not used, the first MTP found is used).

5.8. *IPBX page*

This page allows configuring enterprise IPBX features.

Required:
The IPBX mode must be enabled within “General Parameters” page.
At least a local MTP must be enabled for each call.
The users’ phones can be SIP or H323 phones.
The VoIP/PSTN gateway can be either SIP or H323.

5.8.1. IPBX DTMF commands and audio files

Call Intercept	*01
New Call	*02
Call Conference	*03
Call Hold/Retrieve	*04
Party Line Clear	*05
Call Transfer	*06
Call Forward	*07
Shared line	*08
Eaves dropping	
Hold Music	lounge001_multimax_mono_alaw.sw
Transfer/Refer mode	Proxy
Forward/MovedTmp mode	Proxy
Redirect IVR extension	5999

- *Call Intercept* DTMF sequence to catch a call that rings on another phone that belongs to your group (see Group within Endpoint Profiles page (§ 5.4.2).

- *New Call* DTMF sequence to get a new line during an existing call. The actual remote phone is in hold (with music) and you are invited to dial the number of the new remote phone (terminated with # key).
  You can manage until 4 lines at the same time and switch between them with #1, #2, #3, #4 sequences.
  The New Call sequence is the first step to transfer a call or to create a conference.

- *Call Conference* DTMF sequence to create a conference call with you and 2 other lines. The conference is built with you plus the active line and the 1st line on-hold.
  Note : For advanced conferences (more than 3 participants, meet-me, video, recording…) consult M2Msoft for C3000 conference bridge.

- *Call Hold/Retrieve* DTMF sequence to Hold a call and let it playing music, and to Retrieve (with same sequence) to reconnect.

- *Party Line Clear* DTMF sequence to release the call on active line without hang up and to allow to retrieve another line in hold with #1, #2 sequence.

- *Call Transfer* DTMF sequence to transfer a call (from active line to 1st on-hold).
  The call transfer is also processed when we directly hang up after taking a new call and dial out before or after the remote phone answer call (supervised or blind call transfer).
  This DTMF sequence is necessary if you want to transfer a call after some line switching (#1 / #2).

- *Call Forward* DTMF sequence to transfer a call (from active line to 1st on-hold).

- *Shared Line*
- EavesDropping
- *Hold Music* Global "Music On Hold" file selection.

- *Transfer/Refer mode* Transparent: Refer messages are forwarded to remote phones
  Proxy: Refer messages are managed within S5000.

  NOTE:
  Use Transparent mode when you known that your involved equipments support the REFER sip requests.
  Use Transparent mode when you want to work over SIP trunks in G723.1, or G729. (not G711).
  In all other cases, use Proxy mode.

- *Fwd/Moved tmp mode*

- *Redirect IVR extension* Internal IVR extension to enable/disable a redirect rule.

NOTE: The iPBX commands defined below are entirely managed within the S5000 and are available over any telephone (IP or thru PSTN or analogue systems), any endpoint system.
Furthermore, as more and more SIP telephones provides special contextual keys functions (with on screen display) for:
- transfer
- call hold
- 3 parties conference

All these features are also supported by the S5000 and at the same time as the fully controlled commands below.
The S5000 supports iPBX functions over complex environment with IP and non IP endpoints, complex and basics endpoints.

5.8.2. DTMF/IPBX commands disabling rules

Name	Source	Dest	
FromExtern_ToPhone	0[pa]*	51??	
To_Internal_SVI	*	600*	

This table allows to prevent the DTMF processing for IPBX commands (transfers, conferences…). When the DTMF matches any entry of this table the message is simply forwarded without interpretation. The Source and Destination fields match the direction of the DTMF signal (but not the direction of the call setup).

5.8.3. Redirect rules

You can create up to 20 Redirect rules with criteria to redirect calls to your offices.
The criteria are:

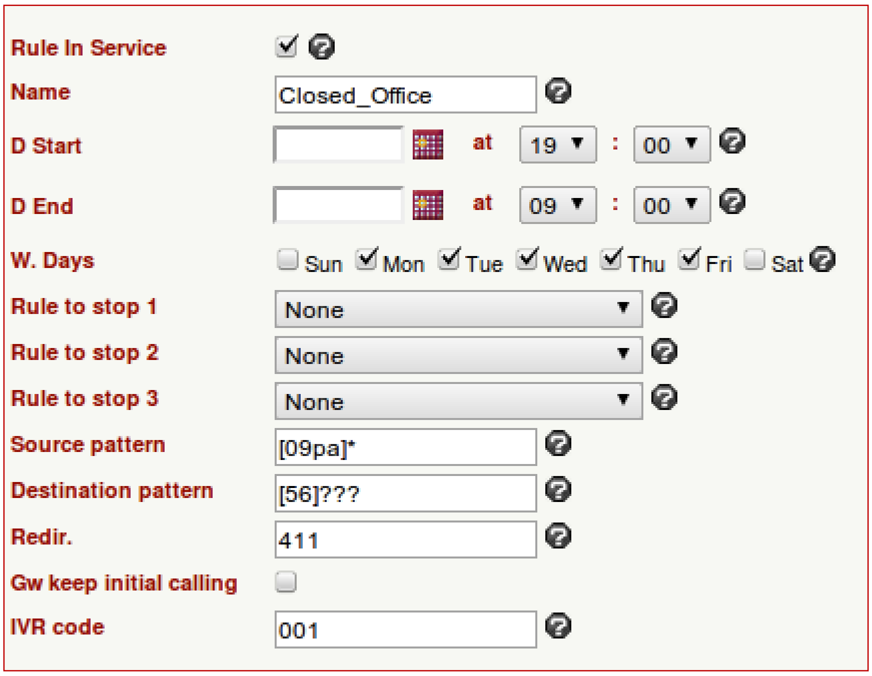
- Period of dates (empty means any date)
- Day of week (select Monday to Sunday)
- Time range (start hour and minutes, end hour and minutes)
- Source pattern (a call from where)
- Destination pattern (a call to where)

It is useful to redirect the call on an audio prompt, or an outside number, during closed office times or days.
For example it is possible to redirect calls for operator for these 4 conditions:

- lunch time on working days,
- nights
- weekends
- days out of office

A redirection can be checked and forced by calling a vocal service (IVR) within S5000 to activate/deactivate a redirection. The automatic cycle will be re used at the next transition of the rule.

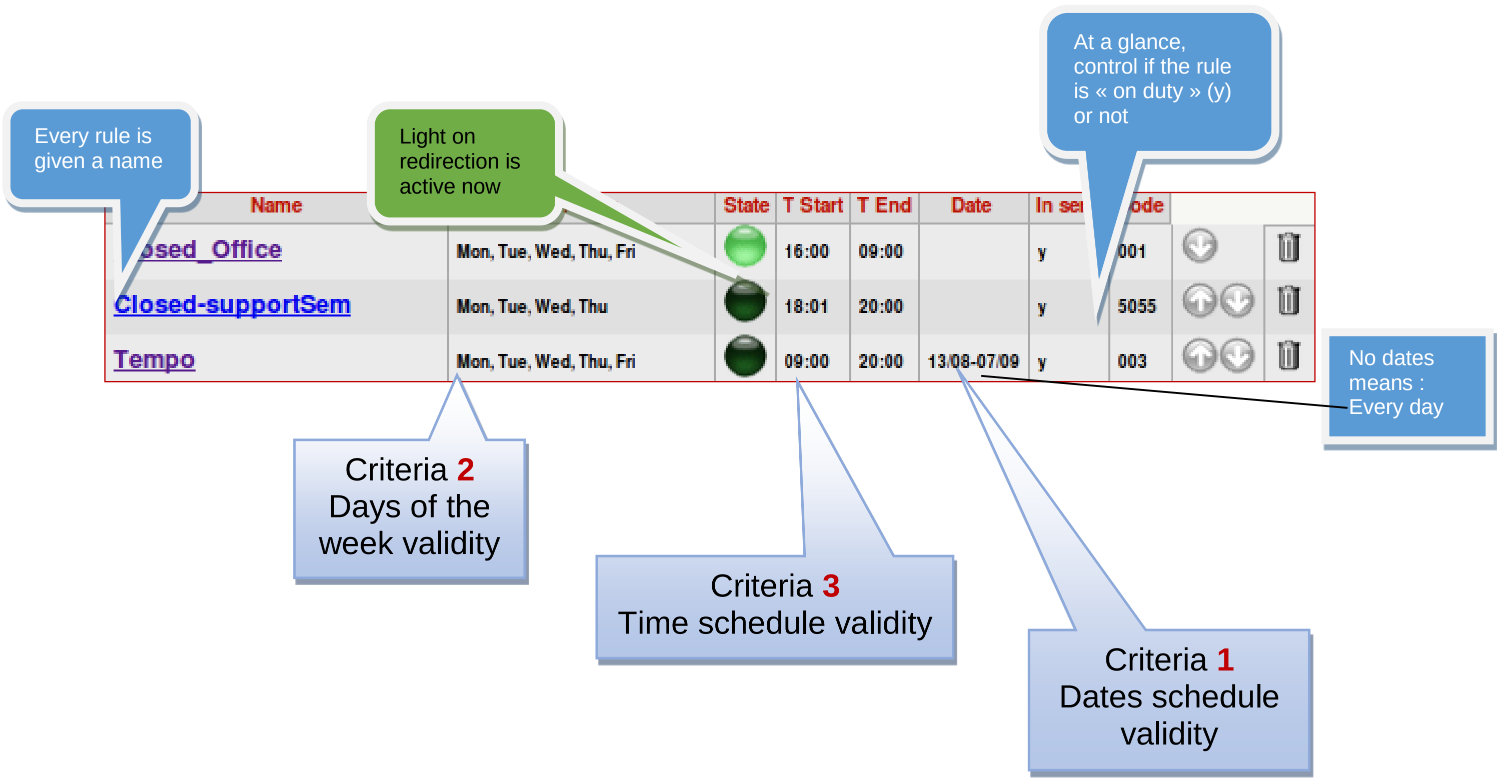
Name	W. Days	State	T Start	T End	Date	In serv.	Code		
Closed_Office	Mon, Tue, Wed, Thu, Fri		19:00	09:00		y	001		
Closed-supportSem	Mon, Tue, Wed, Thu		18:01	20:00		y	5055		
Closed-supportVen	Fri		17:01	20:00		y			



- *Rule In Service* *When checked, means the scheduler is activated on this rule: the rule will be activated and deactivated when time (date and hours) comes.*
- *Name* Rule name.
- *D. Start* Date start of redirection. Format is MM/DD, MM is month, DD is day of the month (example: 05/12 means may – 12th)
- *D End* Date end of redirection. (Idem as previous).
- *W. Days* Days of week to enable redirection.
- Rule to stop (1~3) Selected rule to be avoid when this one is to be active.
- *Src Pattern* Pattern to match source alias of call (i.e. [0p*] to match PSTN or private).
- *Dst Pattern* Pattern to match destination alias of call (i.e. operator of enterprise).
- *Redir.* Forwarded destination alias. It can be a local Media Entity (see § 5.7.1).
- *Gw keep init. calling* Checked to keep initial calling number when call is forwarded again to PSTN.
- *IVR code* Alias of internal vocal service for this rule. The IVR allows to force the status (ON or OFF) of a redirection. The normal cycle will be automatically re used at the next rule transition.

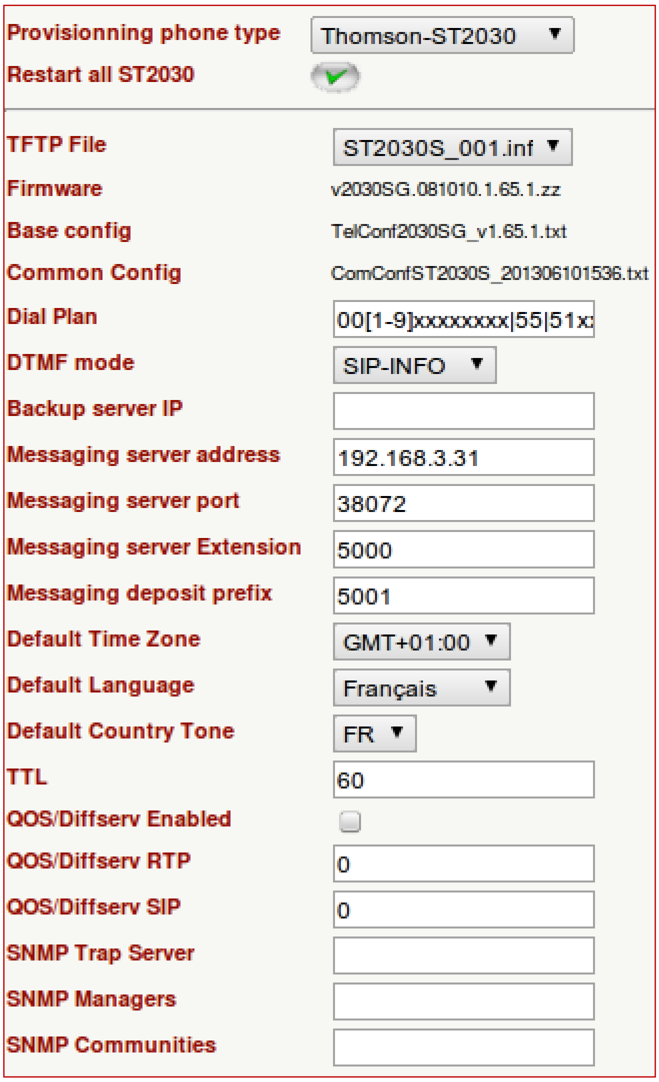
How is it scheduled?

1/ The date range is checked for the rule (no values means it is valid)
2/ If previous is valid, the week of day is checked
3/ If previous is valid, the hours and minutes are checked



5.8.4. Global Auto-provisioning

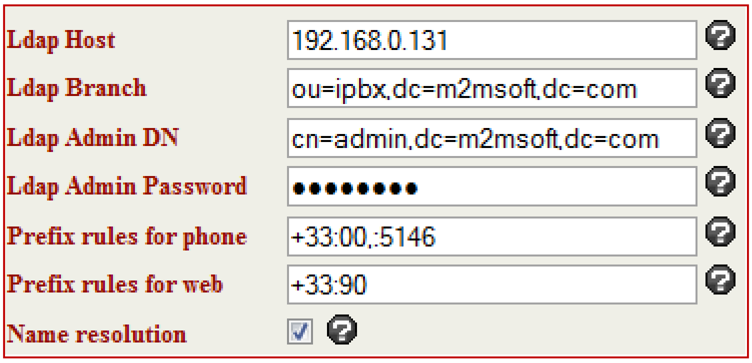
In IPBX mode the Thomson-ST2030 IP-Phones configurations are managed from S5000 (see § 5.4.3). This section describes the common parameters for IP-Phones auto-provisioning.



- *ST2030 TFTP File.* Boot file selection within s5000/tftp directory.
- *Dial Plan* Mask to define the digit collected before the IP-Phone forward call.
- *DTMF mode* SIP-INFO (SIP message) or RFC-2833 (RTP EVENT message).
- *Backup server IP* IP address of backup s5000 (if exists).
- *Messaging server address* Messaging server IP address (for Message Waiting Indicator).
- *Messaging server port* Messaging server SIP port.
- *Messaging server extension* Messaging consultation service extension.
- *Messaging deposit prefix* Prefix of messaging deposit service
- *Default Time Zone* Global devices time zone (can be personalized on endpoint page).
- *Default Language* Global devices language (can be personalized on endpoint page).
- *Default Country Tone* Global devices country tone (can be personalized on ept page).
- *TTL* TTL of all Thomson-ST2030.
- *Restart all Thomson-ST2030* Restart and Re-provisioning of all Thomson- ST2030.

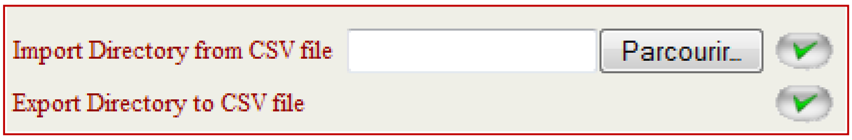
5.8.5. Enterprise Directory administration

LDAP parameters



- *LDAP Host* IP and port server address (x.x.x.x:port). Default port=389
- *LDAP Branch* Sub tree of user entries
- *LDAP Admin DN* Distinguished Name of administrator account.
- *LDAP Admin Pwd* Administrator account's password.
- *Prefix rules / phone* Prefix translations for calls dialled from phone (before:after,before:after,...)
- *Prefix rules /web* Prefix translations for click2talk from web (before:after,before:after,...)
- *Name resolution* To display caller name on phone screen if known in directory.

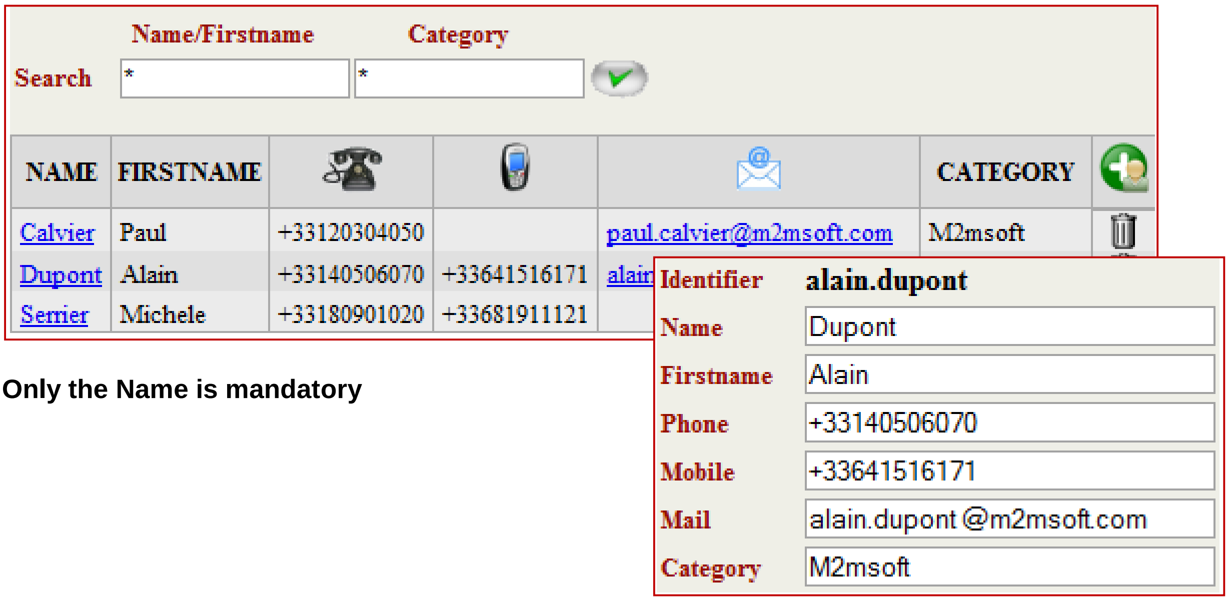
Import/Export directory entries



- *Import Directory* To transfer user entries from CSV file to the directory.
- *Export Directory* To transfer user entries from directory to CSV file.

(To get the field contents export first).

Entries management



Only the Name is mandatory

5.9. Applications page

M2Msoft releases C and JAVA APIs (GKXAPI / JGKXAPI) to build and run users applications that take control of the calls and can authorize, reject, modify and control the communication at any stage of a call.
For more information about the GKXAPI / JGKXAPI, please see API chapter 7.

Any number of different applications, acting on different terminals events or called numbers can be started and connected to the S5000.

The application view shows all the connected applications and their associated contexts states.
A context state is a connection slot that the applications reserves.
Every application handles a number of simultaneous calls on selected conditions (for example, all calls to number 911). When several applications listen the very same set of conditions, they work in a round robin mode: the S5000 forward the calls in a cyclic manner to the different applications: this allow for a distributed and robust service.

A connection slot is either waiting for call or connected. The green light is bright when connected.

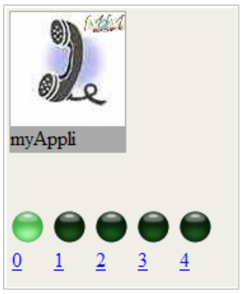


Fig.28 Applications view with connected/waiting slots

NOTE: the number of allowed simultaneous calls can vary according to your license.

Slots inspection

Every application is working with connection slots. A Slot is an application context within the S5000. Every slot is either connected or unconnected, showing a bright green led (connected) or a switched off led (unconnected call).
By selecting a slot, a pop up window opens and refreshes periodically with the following informations:

- living call objects in the slot
  - call objects are : CallWait, ClearWait, OLCWait (for all audio/video channels, for both call parties), ConnectWait
- information associated with the object: state (started: the object is waiting an event ; stopped: the object cannot take a new event anymore) and a value (RTP parameters, called number)

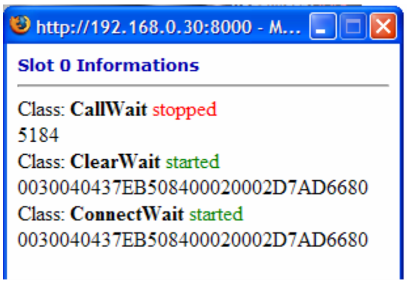


Fig.29 Application slot inspection

5.10. Logs page

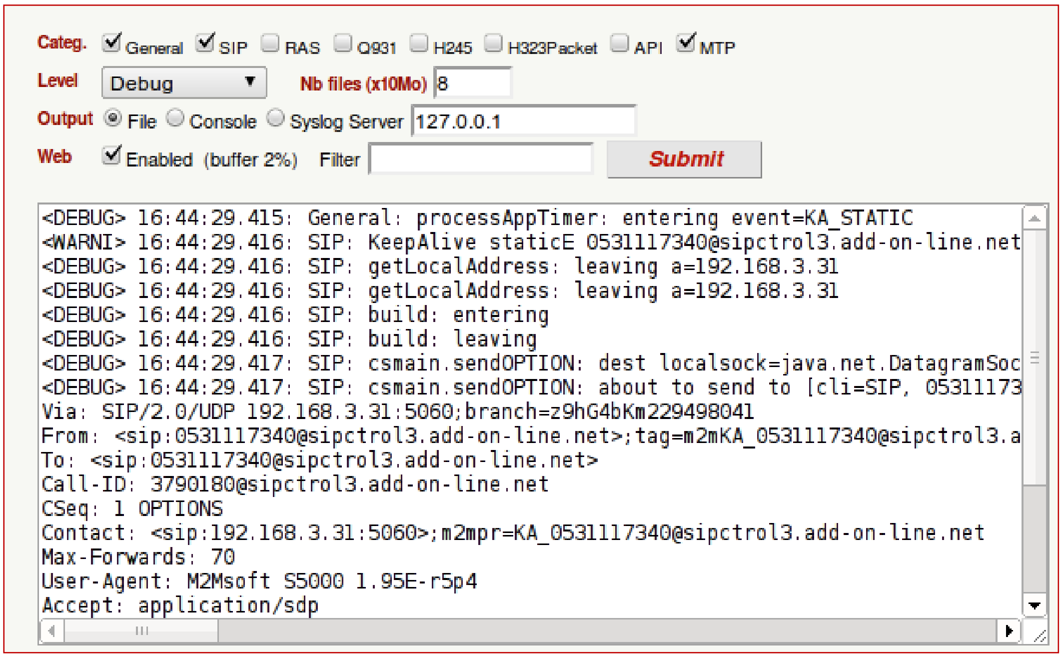


Fig.30 S5000 logs view

- *Categ.* Enable categories to be logged.
- *Level* Minimum Syslog level logged among
  Emergency, Alert, Critical, Error, Warning, Notice, Informational, Debug.
- *Nb files* Number of log files stored (if Output=File). 1 file sizes 10Mo.
- *Output* Output syslog selection :File (s5klog), Console, Syslog Server.
- *Web* Enable a temporary copy toward web buffer (60 Kbytes). A filter can be applied to this buffer. When the buffer is full a “Clear” command is proposed.

Output Syslog server Configuration

Install daemon rsyslog: sudo apt-get install rsyslog
Edit the configuration: sudo vi /etc/rsyslog.conf
Delete comments (#) of those 2 lines:

$ModLoad imudp
$UDPServerRun 514

Comment those 2 lines:

#$PrivDropToUser syslog
#$PrivDropToGroup syslog

Edit configuration: sudo vi /etc/rsyslog.d/50-default.conf

At the beginning of the file add:

$template DynFile,"/var/log/m2msoft/%$year%%$month%%$day%.log"
:fromhost-ip, isequal, "127.0.0.1" ?DynFile
:fromhost-ip, isequal, "127.0.0.1" ~

Create an m2msoft directory for log:

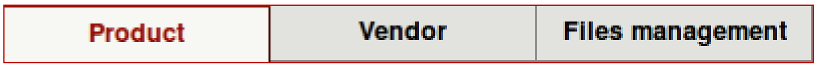
sudo mkdir /var/log/m2msoft
sudo chown syslog:adm /var/log/m2msoft/

Restart syslog daemon: sudo service rsyslog restart

In this example, logs will be redirected to daily files in sub-directory /var/log/m2msoft and from local host (127.0.0.1).

5.11. *About page*

'About' page is subdivided in several sections selectable with tabs.



This page displays information about Product and Vendor. It provides software and configuration update tools.

The following sub-chapters describe all the features.

5.11.1. Product

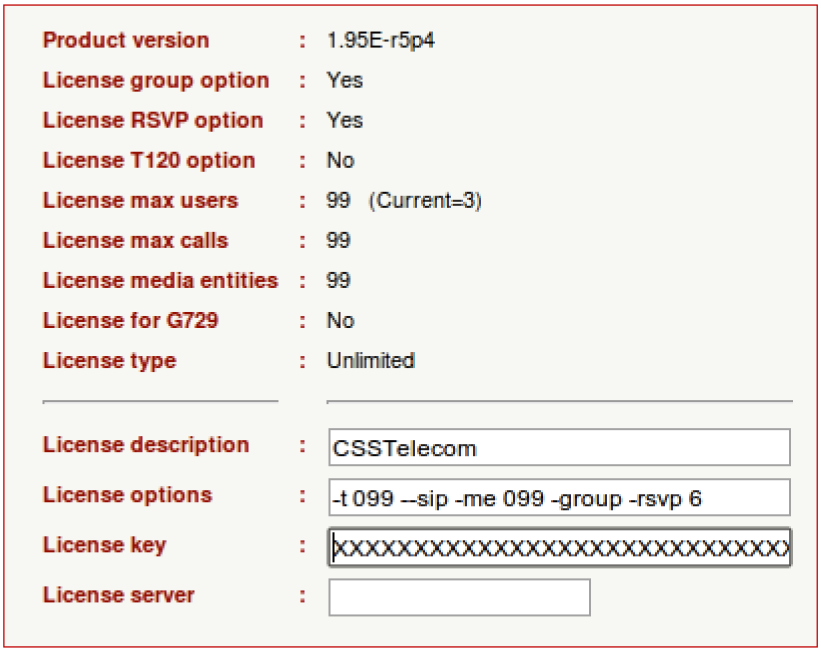


Fig.31 S5000 Product information (version, license, etc.)

- *Description* A description associated with this S5000.
  Can be a mandatory one in case of temporary licenses or a user defined information string.
- *License options* The options string given with your license key. Do not change this line unless you change of license key.
- *License key* The key written within lic.txt file

5.11.2. Vendor

Fig.32 S5000 Vendor information (contact)

5.11.3. Updates

This area allows to manage files transfer (upload and download) between S5000 and PC.

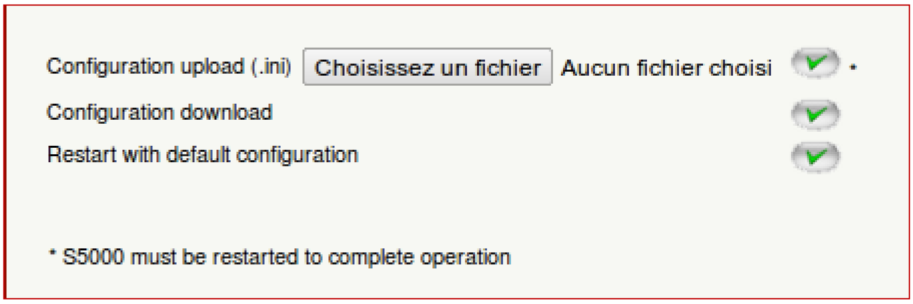


Fig.33 S5000 Update page

- *Configuration updload(.ini)* Step 1: Select you gk.ini file, A popup window opens for you to get locally the gk.ini file. Only select gk.ini file previously saved from the 'Configuration download' option.
  Step 2: click ***upload*** button, the S5000 restarts automatically to take care of the new file.
- *Configuration download* This file is to be kept to be restored in case of restore action on the S5000 server.

6. S5000 Advanced technologies and configuration guide

6.1. SIP interface

According to a license option key, the S5000 is able to handle SIP terminals.
The S5000 supports the following:

- RFC3261 for general operation
- RFC2976 for INFO
- RFC2833 for DTMF thru RTP
- RFC2617 for Digest authentication
- RFC2243 for TLS 1.0
- RFC3265 for subscribe/notify extensions
- RFC2543/3261 (SIP basic call, statefull)
- RFC3581 (rport)
- RFC2833 (DTMF in band avec payload RTP special)
- RFC2246 (TLS 1.0)
- RFC3515 (REFER)
- RFC3428 (MESSAGE)
- RFC3325 (P-ASSERTED et P-Preferred)
- RFC3323 (Anonymisation)
- RFC3311 (UPDATE)
- RFC4028 (Session Expires)
- RFC3407 (Sdp session direct codecs description)
- RFC3420 (SipFrag and sip messages transport within sip notify)
- RFC3264 (dynamic media management within sdp – hold/retrieve)
- RFC4916 (change display/from capability while on call)

6.1.1. Registrar Server

The S5000 acts as a SIP registrar server for the following modes:

- UDP unicast
- UDP multicast (discovery)
- TCP
- TLS

It handles the registration (name, domain, ip address, time to live, etc) of SIP terminals that want to have a central system to find the users and support services for them.

> NOTE: **Be sure to have set a valid domain** name or ip address for your S5000 to work properly. *(see chapter 5.3.2)*

Handling unregistered sip terminals

Unregistered SIP terminals can be reached and can emit direct invite calls within the S5000: the S5000 automatically defines a static listener for external (unregistered) endpoints.
The embeddedService/Routes can be used to let H323 or SIP endpoints access to unregistered SIP terminals in the same way than registered ones.

6.1.2. Proxy Server

The S5000 acts as a proxy server. It handles direct calls to the SIP or H323 parties directly.
When a registered user dials a SIP url (alias@domain), the server operates a search for the party.
Search is made as follow:
Search through the embeddedServices rules
If no rules are found:
stepB processing

- search amongst the registered SIP users within the S5000
  - If the party is found (endpoint or route), then the call is directed to the party
  - if the party is not found :
    - search amongst the H323 registered endpoints (using the default domain)
      - if the party is found, the call is directed to the H323 party
      - if the party is not found :
        - if the SIP domain is an IP address, then the call is given directly to the party IP.
        - if the SIP domain is a host name, then a DNS search is operated on the S5000
          - if found, the call is directed to the IP address found
          - else, the call is rejected.
          -

If a rule has been found, modify calling and called party as defined within the embedded service, then process the target/route resolution as step B.

NOTE: Please consult the release notes for the list of the supported SIP terminals, features and APIs.

6.1.3. SIP/H323 Gateway

The S5000 acts as a SIP to H323 and H323 to SIP Gateway.
Endpoints from both protocols can register and call together.
Incoming H323 calls can be transformed into SIP calls and reverse.

Using a S5000 as a frontal SIP access enables H323 systems to be called from SIP terminals.

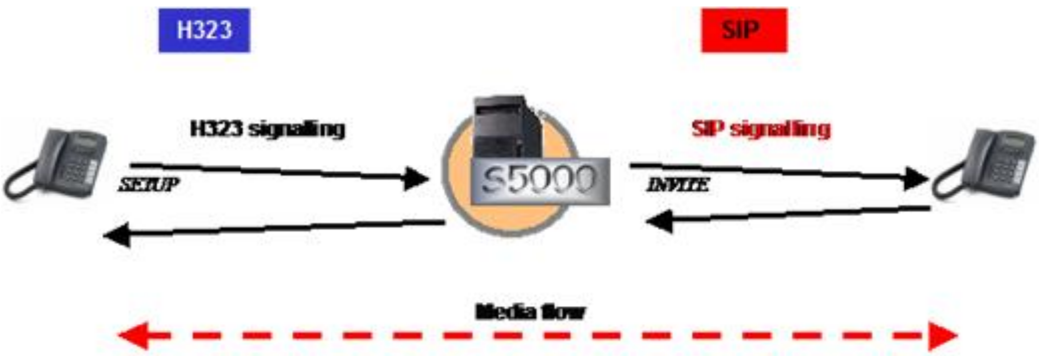


Fig.35 S5000 H323/SIP gateway engine

H323 to SIP call works as follow :

- search amongst the embedded services for the called number. A SIP redirect can be found at this stage (usefull to reach unregistered SIP recipients) (see § 5.6 for embedded services)
- If no H323 party no SIP redirect has been found, then a search amongst SIP registered endpoints is made
  - if the SIP party is not found in registered terminals, a releaseComplete is returned to the caller
  - else Invite to the recipient and SIP/H323 signalling takes place

SIP to H323 call works as follow:

- search amongst the registered SIP endpoints or sip uri analysis to reach unregistered endpoints
- If no SIP party has been found, then a search amongst H323 registered endpoints is made
  - if the H323 party is not found in registered terminals, a BYE is returned to the caller
  - else Setup sent to the recipient and SIP/H323 signalling takes place

NOTE on media paths
Due to the very different signalling protocols, the S5000 SIP/H323 internal Gateway makes its best to have the media flows going directly between the H323 and SIP endpoints in each conversation way. While processing the call, some flows might go through the S5000 temporarily. Some SIP terminals may not accept to have their media streams redirected at a point in call process and keep a channel through the S5000.

6.2. *Media Entities*

The S5000 can embed a number of media resources without any need for an external media server. The Media Entities are shared resources within the S5000 that allow to play (sound) files to endpoints upon connection. This works in dial in calls as well as dial out calls.

6.2.1. What for?

A Media Entity is a convenient way to direct your callers to a specific announcement (dial in mode). It can also be used to send some voice message to someone (dial out mode).
The S5000 can connect two media entities and thus connect two endpoints after some waiting announcement.
Typical applications are:

- Voice answering machine (redirect the calls to a Media Entity whenever you're not available).
- Status message (coupled with an S5000 application, to terminate a special number dialin).
- Voice alert (coupled with an S5000 application, dial out a voice message to someone phone)
- Call Center (let's wait incoming calls and connect incoming numbers to local operators' phones as they are available: this can be done through the web browser or automatically with an S5000 application).

A Media Entity is associated with an alias, a file name, codec list and a loop mode.
The alias is the way to identify and route calls to it. The file name is the voice file to play to callers or called. The loop defines if the Media Entity needs to play a number of times then releasing the call. The codecs list defines the available audio codecs to be negotiated with parties. The associated files must be present on disc at play time for the different negotiated codecs.

6.2.2. Usage

Simply define the number of simultaneous voice capabilities you need.
Note that your license rights may limit this number.
Configure Media Entities (see § 5.7.1).

In the Endpoints view (see § 5.4.1), your media entities appear as:

Alias ▼▲	Sig. ▼▲	Type ▼▲	Address ▼▲	TTL ▼▲	Grp ▼▲	Superv ▼▲	
99001		MEDIA					connectTo
911		MEDIA					connectTo
912		MEDIA					connectTo

Dialin mode
Any user can call the Media Entity numbers (for example 911) and the associated voice/music message will be played to him.

Dialout mode
With the Browser, one can make a dialout call by clicking "connectTo" link and choosing a recipient to call (see below). All the available aliases (registered endpoints) are shown as well as a "free form" number. This can be used to call through an external system: for example a PSTN number through a gateway or a Cisco's Call Manager or an Alcatel iPBX.

Fig.36 S5000 Media Entity dial out choice

The media Entity dials the number. After ringing and connection, the dial out user hear the voice/music message associated with the selected Media Entity.

Join calls

With a number of connected calls with the media entities channels, the S5000 allows you to bind any of these calls through the web browser interface (see § 5.5.1) or with the S5000 API.

STATE	CALLING	CALLED	CALLID	EXTRA	
[Connected]	5183(0)	911 911(0) PLAYING	222319AB49AF1B7F346962C93FBF364C		disconnect join
[Connected]	5188(0)	912 912(0) PLAYING	0030040511913C7A0002000001650E40		disconnect join

Two “half calls” are shown.
To connect them together, just select one on them join link.
A new window opens with the list of the other available Media Entities.
Choose one of them and validate.

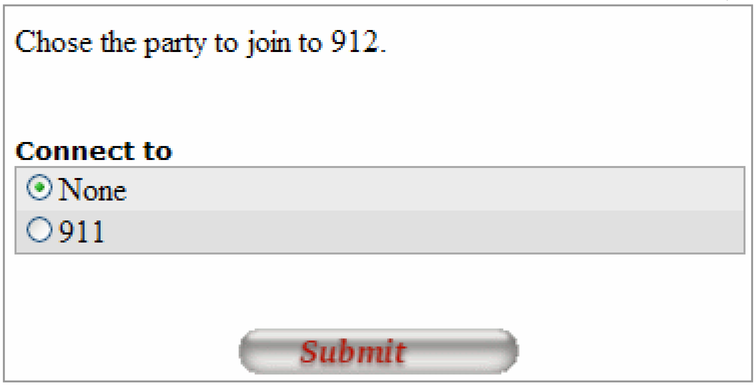


Fig.37 S5000 Media Entity join choice

Immediately, the two “half calls” are joined and the users talk together. The Media Entity status is now “JOINED” within the conferences display (§ **Erreur ! Source du renvoi introuvable.**).

conf_0	[Connected]	5188(0)	912 912(0) JOINED	0030040511913C7A0002000001650E40		disconnect play
	[Connected]	5183(0)	911 911(0) JOINED	222319AB49AF1B7F346962C93FBF364C		disconnect play

Disjoin calls
As soon as calls are joined, a “play” link appears that allow restoring the initial state.
The half calls are no longer joined and the users listen again their announcements.

6.2.3. Limitations

The media entities can be connected to one endpoint at a time. You may declare as many Media Entity as you need (your license rights can limit this).
The supported files formats are:

- G711 Alaw 30 ms
- G723.1 30 ms
- G729 20 ms

6.2.4. Use with application programming interface

The S5000 API allows to handle Media Entity in numbers to build professional services such as:

- Conference server, connecting 3 or more Media Entities and performing RTP mixing
- Call Center, handling a number of waiting calls and internal -operators- calls to be joined
- Voice mail, to redirect a caller to a voice message then taking the recorded file and having it emailed to the recipient
- Protocol or network gateway, to use media Entity as media switch between different network types
- Etc…

6.3. *Media Termination Points (MTP)*

The S5000 is a softswitch which routes signaling flows (H225, H245, SIP). The S5000 can also route media (RTP) flows.
This feature is more than useful for operators who want mask their customers IP addresses or used for enterprises extended services such as Hold, Transfers, call parks, etc.…

The following figure shows 2 MTP modules managed by a centralized S5000:

- A local MTP (near the S5000),
- A remote MTP.

In this example, a call between 2 users within the same area use the remote MTP (also within the same area). But a call between 2 separated areas use the local MTP.
Thus bandwidth is optimized over Wan links.

Some rules must be configured to associate an MTP (or not) to a call. They are based on source and destination patterns.

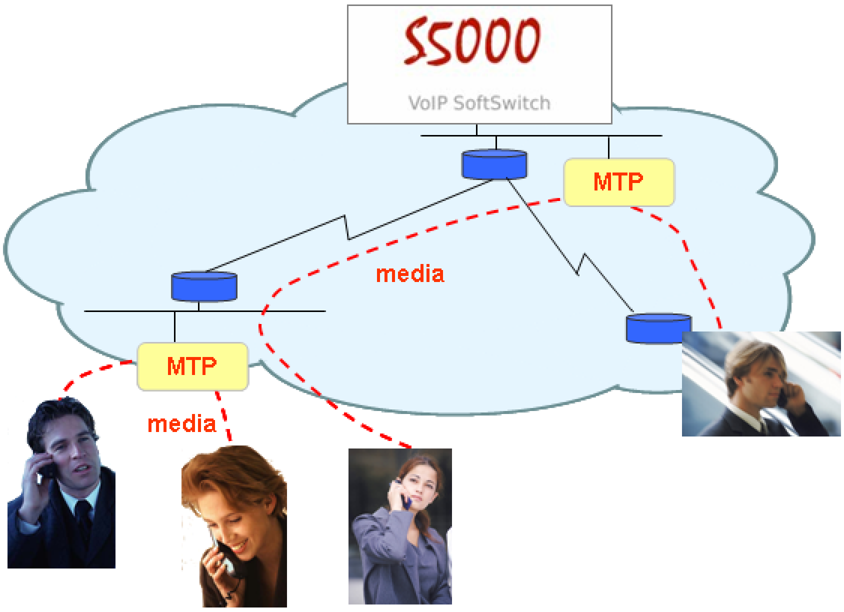


Fig.38 S5000 Media Termination Points overview

The S5000 opens a TCP control connection with every defined MTP and sends the commands to open and close RTP channels with associated UDP ports, over this connection.

When an MTP is defined within the configuration (see § 5.7.2), the TCP control connection is attempted from S5000 toward MTP.
If this connection is established, the status green LED lights and the number of available channels is displayed. Otherwise the LED is off.

In this example the "mtp0" MTP is connected and the "Paris" MTP is not.

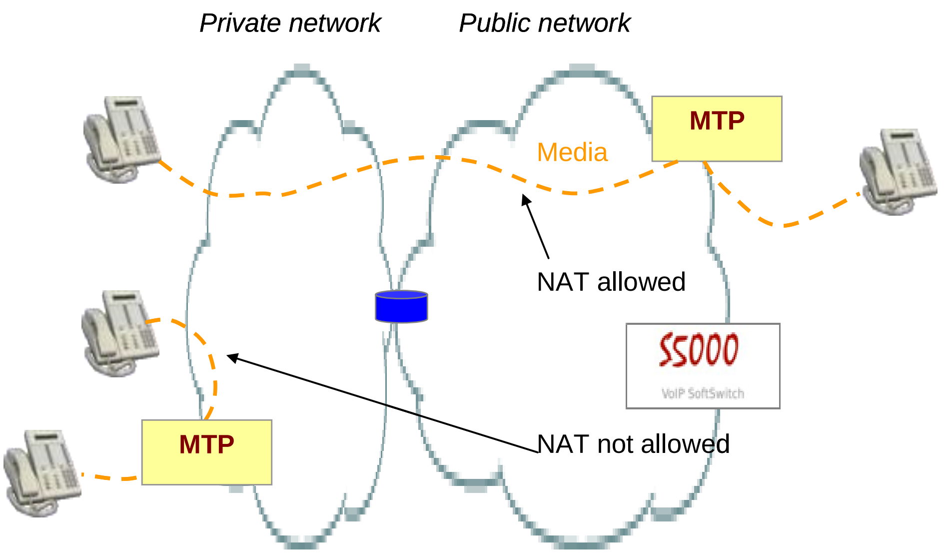
Name	IP address	TCP Port	1ST RTP	State		Current	Version	
mtp0	192.168.3.31	29000	29000			0/100	1.126	
Paris	192.168.3.30	30000	30000			0/0	?	

Local MTP provides 10 RTP channels. When a call uses an MTP, 2 channels are busy for it, one for each remote endpoint. Thus 10 channels allow for 5 simultaneous calls.

Each channel uses 2 UDP ports: one (even value) for RTP and the next one (odd value) for RTCP. Thus in the previous example Paris will be able to bind UDP ports from 28000 until 28019.

Note: When the MTP runs within the same server as S5000, the IP ADDRESS parameter must be the real network IP address and not the loopback address 127.0.0.1.

When remote endpoints within a private network are registered on S5000 within a public network (Internet IP address), the Automatic NAT feature (see § 6.4) will modify the addresses fields into the packets. In the particular case of remote MTP within the private network (as shown in following figure) the automatic NAT must be disabled on MTP to keep the private IP addresses within RTP packets. In this case the ***NAT allowed*** parameter must be set to **NO** in the MTP configuration (see § 5.7.2).



The MTP Rules allows defining in which case a call must use an MTP for media, and which one. These rules are based on source and destination patterns.

Name	Source	Dest	MTP	CH. IN	CH. OUT	
Out to International	*	00*	Local	Auto	Increase from 0	
Rule 1	40*	40*	Paris	Auto	Increase from 0	

In this example, when a call matches destination 00* (for any source), the Local MTP is selected, when a call matches source 40* AND destination 40*, the Paris MTP is selected. In other cases no MTP is selected and media is directly exchanged between endpoints (no hold / transfers…).

Optionally:
- MTP also generates in band DTMF from out of band signals (SIP INFO, RFC2833)
- MTP can operate a crypt/decrypt operation on RTP flows (RFC3711 SRTP)

6.4. Automatic NAT handling

When working with endpoints behind a NAT system – which is a standard mode for residential telephony with subscribers behind a home firewall/router- common problems arise from a private/public mismatch addresses.

Users behind a NAT router, assign a private address (example is 192.168.0.a.b) to their terminal or terminals. Especially if they share an IP address amongst various computers.
Simply registering a terminal to an external publicly known S5000 could be difficult as the H323 or SIP information messages that go out of the terminal shows the private IP address.

The S5000 has the ability to automatically detect if a registering terminal is behind a NAT and handle these accordingly to have signalling and media flows going back to the right addresses and ports.

Hence, there is no need of special Router configuration nor special NAT capable terminals to be registered and dialling out calls.
For receiving incoming calls, the residential subscriber has to use DMZ or port forwarding to direct incoming SETUP or INVITE to its terminals or use the S5000 freeVPN solution to handle more than one local endpoint.

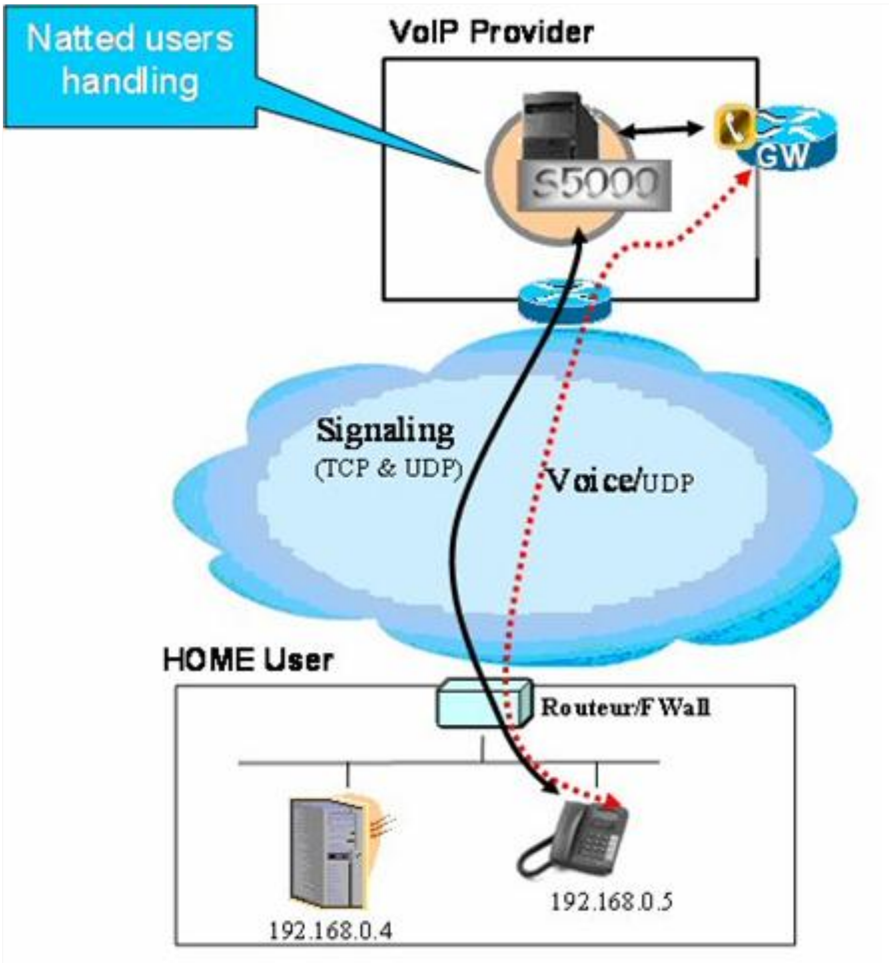


Fig.39 S5000 Automatic NAT configuration case

6.5. Inter-site trough Internet (No VPN solution)

The S5000 allows for effortless inter connection of multiple sites with IP telephony, even when no VPN are available.

Simply install an S5000 in every site, interconnect them through public or private IP links and set a simple configuration. All users can now talk to each other.

The system is called M2MFreeVPN and enables inter site configuration even when no VPN is set or available.

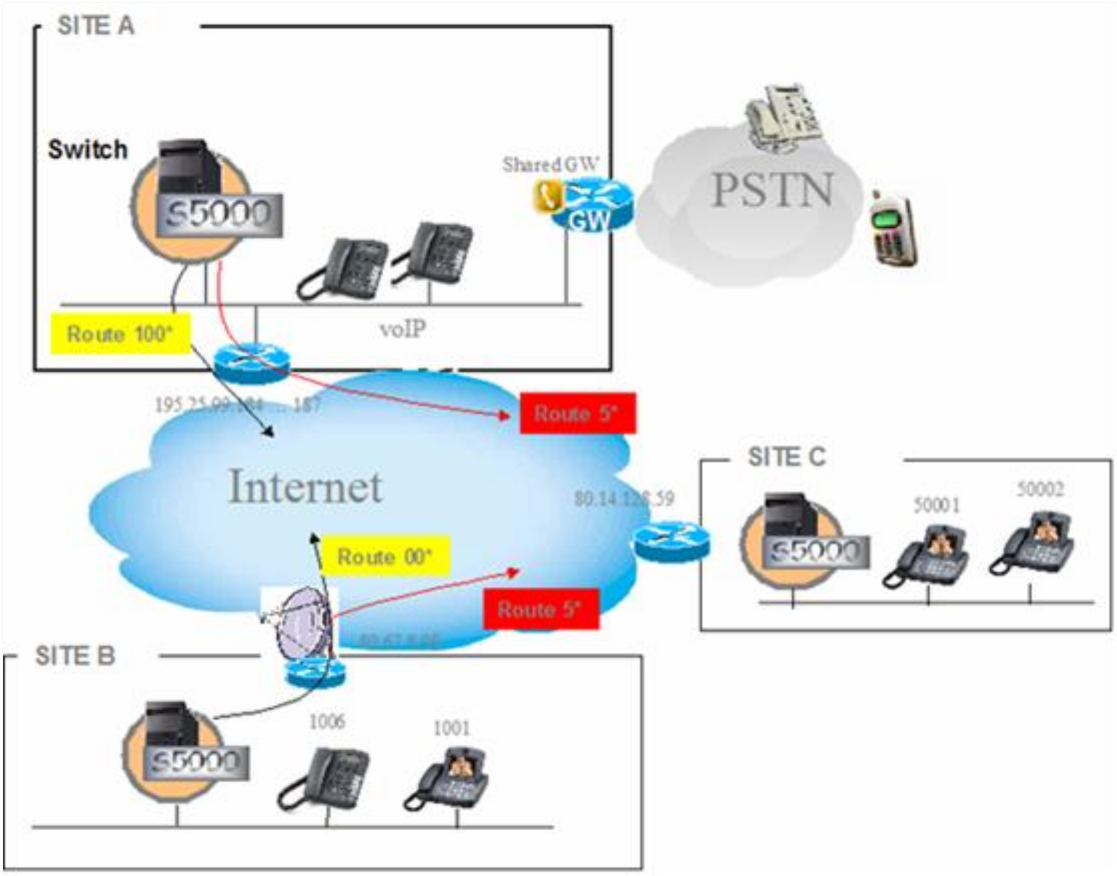
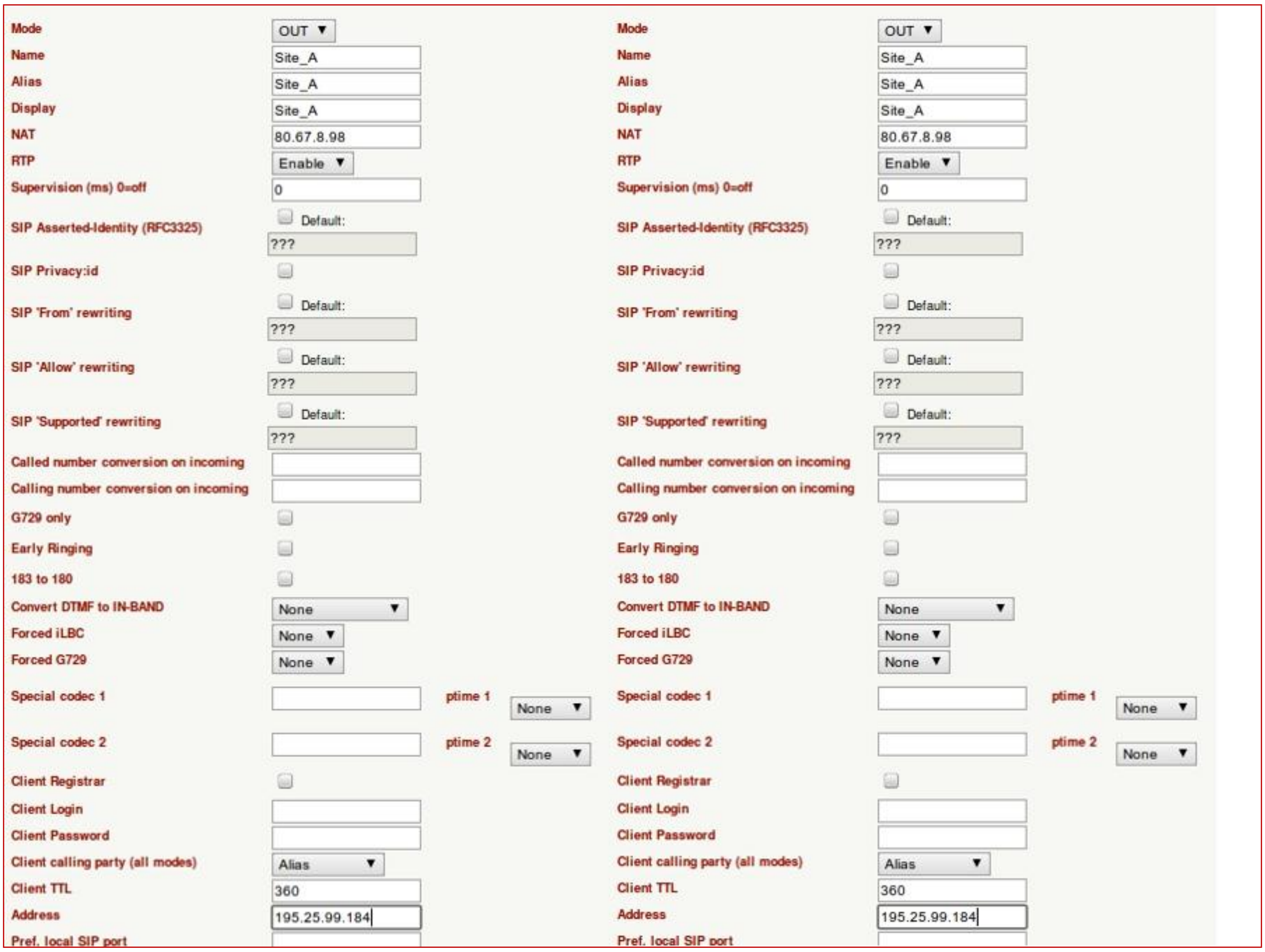


Fig.40 S5000 Automatic 1 public to N private configuration handling (free VPN)

6.5.1. Routing configuration

The following is considered on Site B which must know the routes to reach 0* destinations (Site A) and 5* destinations (Site C).
Create 2 Static Entities (mode OUT) to consider remote S5000 as endpoints, with M2MFreeVPN feature (NAT + RTP translator):



Mode	OUT	Mode	OUT
Name	Site_A	Name	Site_A
Alias	Site_A	Alias	Site_A
Display	Site_A	Display	Site_A
NAT	80.67.8.98	NAT	80.67.8.98
RTP	Enable	RTP	Enable
Supervision (ms) 0=off	0	Supervision (ms) 0=off	0
SIP Asserted-Identity (RFC3325)	Default: ???	SIP Asserted-Identity (RFC3325)	Default: ???
SIP Privacy:id		SIP Privacy:id	
SIP 'From' rewriting	Default: ???	SIP 'From' rewriting	Default: ???
SIP 'Allow' rewriting	Default: ???	SIP 'Allow' rewriting	Default: ???
SIP 'Supported' rewriting	Default: ???	SIP 'Supported' rewriting	Default: ???
Called number conversion on incoming		Called number conversion on incoming	
Calling number conversion on incoming		Calling number conversion on incoming	
G729 only		G729 only	
Early Ringing		Early Ringing	
183 to 180		183 to 180	
Convert DTMF to IN-BAND	None	Convert DTMF to IN-BAND	None
Forced iLBC	None	Forced iLBC	None
Forced G729	None	Forced G729	None
Special codec 1	ptime 1 None	Special codec 1	ptime 1 None
Special codec 2	ptime 2 None	Special codec 2	ptime 2 None
Client Registrar		Client Registrar	
Client Login		Client Login	
Client Password		Client Password	
Client calling party (all modes)	Alias	Client calling party (all modes)	Alias
Client TTL	360	Client TTL	360
Address	195.25.99.184	Address	195.25.99.184
Pref. local SIP port		Pref. local SIP port	

Fig.41 Static entities configuration for a private site connected to public internet (outgoing calls handling)

Create 2 Embedded Services which forward calls to the remote S5000:

Name	To_Site_A	Name	To_Site_C
Type	FORWARD	Type	FORWARD
Sip account mask	*	Sip account mask	*
Source Mask	*	Source Mask	*
Destination Mask	0*	Destination Mask	5*
Sip Codecs mask		Sip Codecs mask	
Media file	-	Media file	-
Forwarded destination	*	Forwarded destination	*
Forwarded source	*	Forwarded source	*
Target	Site_A Application	Target	Site_C Application
Route	None	Route	None

6.5.2. Definition of global listener for external sites

The following is considered on Site B which must accept incoming calls from Site A and Site C. Create 1 Static Entity (mode IN) to enable incoming H225 connections (i.e. port 1720). One can filter the remote IP addresses:

Mode	IN		
Name	globalListen		
Alias	globalListen		
Q931/SIP Listener Port	1720		
RAS Port	1720		
NAT	80.67.8.98		
RTP	Disable		
Supervision (ms) 0=off	0		
Forced G729	None		
Special codec 1		ptime 1	None
Special codec 2		ptime 2	None
Source IP filtering	195.25.99.184		
	80.14.128.59 195.25.99.184		

Fig.42 Static entities configuration for a private site connected to public internet (incoming call handling)

6.5.3. Router/Firewall settings

In order for the system to work you must set some internal NAT/PAT rules on your internet connected router.
IP NAT for the outgoing packets must also be set within the router.

The used TCP and UDP ports are configured within *General Parameters* (see § 5.3.1).

Q931/H245 Ports Range	10000-14999
RTP Ports Range	10000-14999

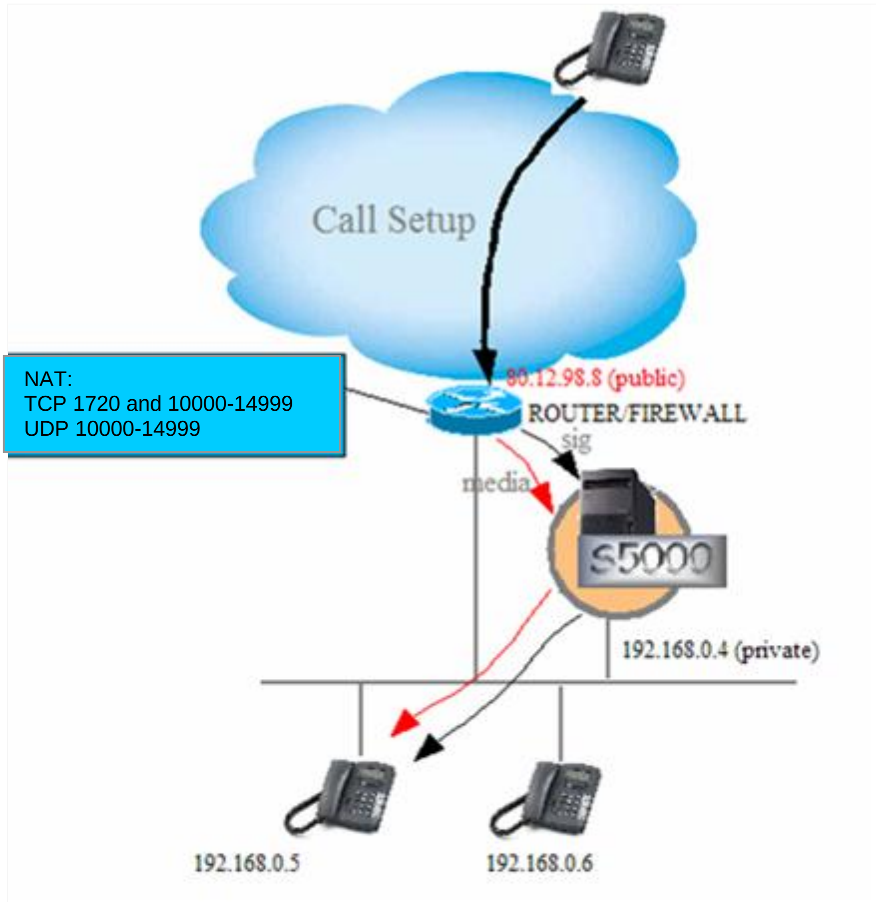


Fig.43 Cross over firewall S5000

On Router configure:
External TCP 1720-1720 → Internal TCP 192.168.0.4
External TCP 10000-14999 → Internal TCP 192.168.0.4
External UDP 10000-14999 → Internal UDP 192.168.0.4

6.6. *Endpoints registration control*

It could be useful for security reasons to select which endpoints are able to register the system. This control from the S5000 embedded services is available for all H323 and SIP terminals.

The RREJECT and RACCEPT type embedded services are used for that matter.

6.6.1. Reject a set of endpoints

To reject a set of endpoints with some specific alias or aliases range, just add an embedded service of ***type***=RREJECT with ***Destination mask***=<alias pattern>.

Whenever the mask matches, the terminal registration is rejected by the S5000.
An enhanced precision can be achieved on the productId information. One can **reject selected terminals** belonging from a specified product Identification, for example, 'Microsoft Netmeeting' or 'Hinet LP 5100'.
To specify that only special productId must be rejected, specify ***'vendor***' value within the ***Destination mask*** parameter and the productId string within the ***New Calling Number*** parameter.

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route
Reject_54XX	RREJECT	*	54*	*	*	*
Reject_NetMeeting	RREJECT	*	vendor	*	Microsoft NetMeeting	*

Fig.44 Registration control (open/closed modes)

6.6.2. Accept a set of endpoints

To accept only a set of endpoints with some specific alias or aliases range, just add an embedded service of type RACCEPT with mask= ***Destination mask***=<alias pattern>.

Beware of the rules order! The first match wins.
Note: Default mode is a S5000 in open mode, all endpoints can register.

In the following example terminals matching aliases 51* are accepted, all the other are rejected:

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route
Accpt_51XX	RACCEPT	*	51*	*	*	*
Reject_other	RREJECT	*	*	*	*	*

Fig.45 Registration control (open/closed modes) with some endpoints to accept

6.7. *Advanced Routing*

Every call request entering the S5000 from either protocol is controlled based on different rules.

6.7.1. The simple way

By definition, the simple case is calls between registered endpoints, dialled and dialling numbers are known.
There is no need for special rules or processing, the S5000 has the knowledge of the parties and knows how to route calls between them.
The S5000 maintains permanently the IP address to join registered endpoints even behind a NAT/Router.

Example:
Let's defines two IP telephones (SIP or H323) with user number 100 and 200.
These phones are shown within the S5000 Web interface on endpoint view.

When 100 dials 200, the 200'endpoints will ring and connect.

6.7.2. The advanced way

The simple point to point call between registered endpoint is not always suitable, for enterprise or carrier services, more complex routing rules must be handled.

The routing can be processed by Embedded Services, by external Application, and also both.

An Embedded Service can be used to forward a call to a single destination (Target), or to a Route which can try several destinations in cases of busy, noAnswer, unavailable destination, etc…

A call select an Embedded Service according Source AND/OR destination patterns.

An Embedded Service can be used to only modify digits within source and/or destination addresses without fixing target recipient.

Example 1: ***(Embedded Service for digit modifications)***
When destination matches 1234*, the 2 first digits must be removed from calling number and prefix 33 added (without change target):

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route
Example1	FORWARD	*	1234*	*	++33*	*

Example 2: ***(Embedded Service forwarding call to single target)***
When source matches 5*, the call is to be forwarded to 5150 (registered SIP or H323 endpoint):

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route	
Example2	FORWARD	5*	*	*	*	@5150	T

***Example 3*: (Embedded Service forwarding call toward a route + Route + Trunks)**
When the Direct Inward Dial (DID) extension matches 4000, the call is to be forwarded to 3 internal users (4001, 4002, 4003) in rotary group mode (Call Center model). If no user is available (busy / no answer / not registered) the call must be forwarded to the voice messaging 4004:

Embedded Service

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route	
Example3	FORWARD	*	4000	*	*	CallCenter	R

Route

Name	Mode	Trunk 1	Trunk 2	Trunk 3	More
CallCenter	BKP	Users	Messaging		

Trunks

Name	Max call	ALGO	Target 1	Target 2	Target 3	More
Users	Unlimited	Rotary	@4001	@4002	@4003	
Messaging	Unlimited	Ffirst	@4004			

***Example 4*: (Embedded Service forwarding call toward an application only)**
When destination matches 6000, the call is to be forwarded to application the named “myApp”:

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route	
Example4	FORWARD	*	6000	*	*	myApp	A

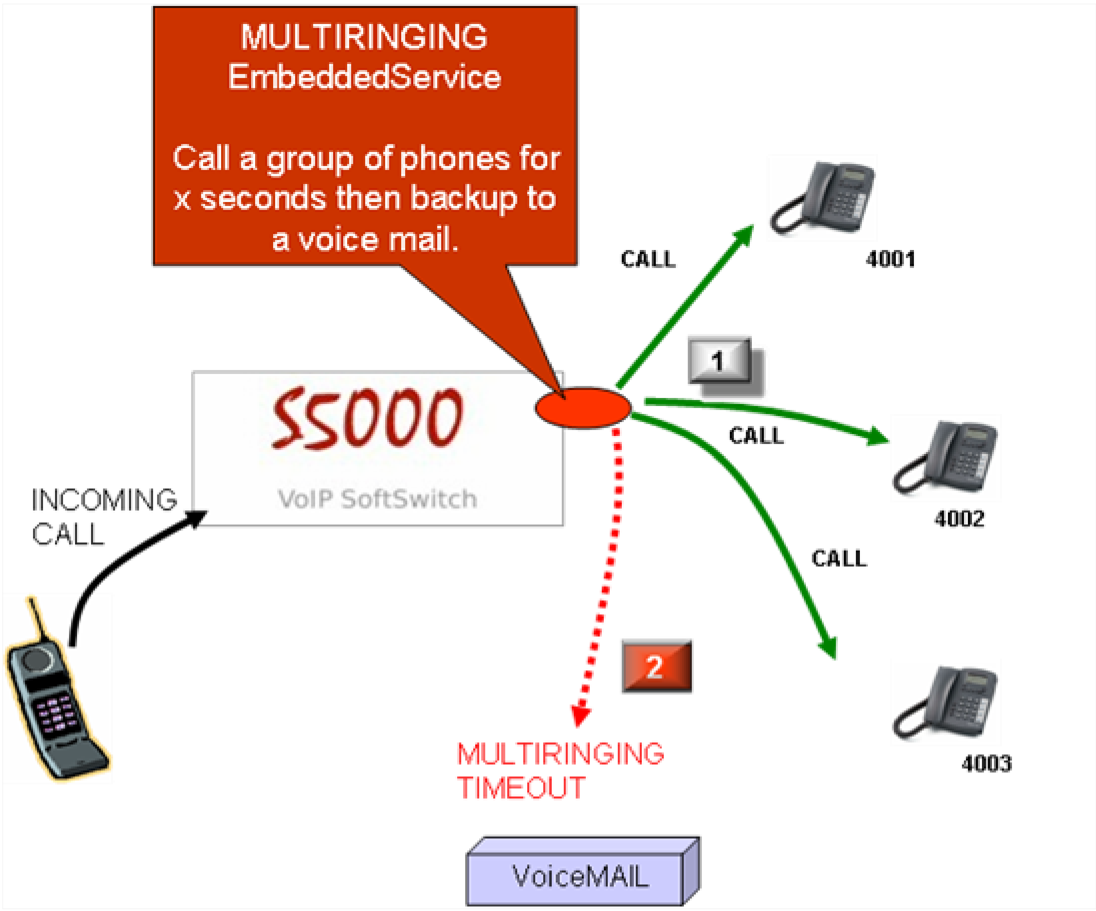
***Example 5*: (Embedded Service forwarding call toward an application and then to a Route)**
When destination matches 70*, the call is to be forwarded to the application named “myCDR”, and then to the Route named “CallCenter”.

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route	
Example5	FORWARD	*	70*	*	*	CallCenter	A+R

Application : myCDR
Route : CallCenter

***Example 6**: (Embedded Service forwarding call to a MultiRinging group of phones, then on no answer terminates the call onto a messaging system)*

When the Direct Inward Dial (DID) extension matches 4000, the call is to be forwarded to (*TRUNK_MR*) 3 internal users (4001, 4002, 4003) in MULTIRING group mode: all three phones will ring simultaneously.
The first user to pick up automatically is connected and the other phones stop ringing.
If no user is available (busy / no answer after the MULTIRING timeout specified/ not registered) the call must be forwarded to the voice messaging 5000 (*Trunk Messaging_M5000*):



Embedded Service

Name	Type	Src Mask	DEST Mask	Fwd Dest	Fwd Src	Target / Route			
Example6	FORWARD	*	4000	*	*	Route_MR	R		

Route

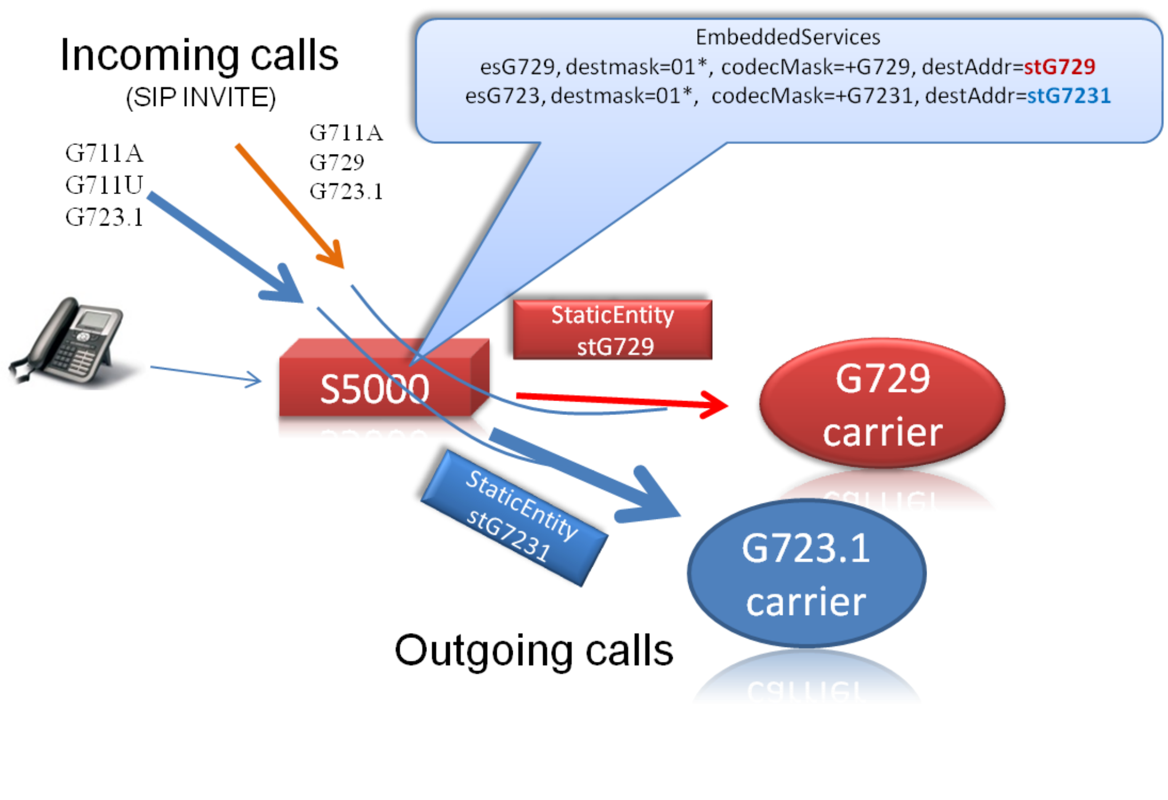
Name	Mode	Trunk 1	Trunk 2	Trunk 3
Route_MR	BKP	Trunk_MR	Messaging_M5000	

Trunks

Name	Max call	ALGO	Target 1	Target 2	Target 3	More	
Trunk_MR	Unlimited	MultiRing	@4001	@4002	@4003		
Messaging_M5000	Unlimited	Ffirst	@5000				

6.7.3. Routing according to CODECs

The S5000 allows to route calls according to calling, called, sip account and optionally audio codecs. Depending on the presence of some codecs, the call can be routed to a specific destination, -for example a carrier that does accept only G729- or another –to a carrier that works in G71A only-. This is an hieved by using EmbeddedService, ROUTE and TRUNK definition for your routing.



In the previous example, one defined two embedded services, one to direct calls that have at least G729 codec towards a G729 termination; the other embedded service direct calls that have at least G723.1 codec to another termination. This avoid codec transcoding and allow calls to establish end to end within the right codec.

6.7.4. Routing according to DNS (SRV, NAPTR, ENUM)

a) Principles

Amongst the many routing scheme that support the S5000 (based on registered aliases, prefixes, suffixes, number patterns, codecs, fixed selection of multi destinations, etc.), the S5000 allows, for SIP calls, to use a 'to' domain analysis through DNS search.
When above described searches failed or if explicitly requested, the S5000 will try to find a SIP proxy server that can handled a recipient merely known as a phone number and a domain name.

This DNS search scheme makes use of SRV and NAPTR DNS entries as defined by RFC3262.

SRV records allow an organization to set a SIP proxy server IP address to handle calls to this organization.
Example: 7868@myco.com.
The direct DNS request (A record) for myco.com gives 193.6.7.8. : this may not be the best and appropriate address to send the INVITE _

SRV DNS request for udp.sip type gives 193.67.7.250 and port 5062: this is better and this is tagged as a SIP server for this organization.

NAPTR records allow to list entries associated with E164 phone numbers.
Here we no longer make use of the recipient domain but the phone number.
Example: 7868@myco.com.
NAPTR DNS search for type : 8.6.8.7.e164.arpa (Nate the phone number digits reversal) gives : sip:johndoe@romaniatel.ro
This is a URI for the user and that can be totally different from the original 'to' URI of the original call.
The recipient 7868@myco.com (example) can then be contacted at this URI sip:jondoe@romaniatel.ro.
A DNS search for SRV record or A record is to be made from that point.

It exists several E164 databases in the world.
e164.arpa, freenum.org, etc.
According to the search domain, this will be added as suffix for the search.
A failed search for 8.6.7.**e164.arpa** can be successful for 8.6.7.**freenum.org**.

b) Use

The DNS search is naturally done within the S5000 when, after passing all routing procedures (registered set, embedded services, application), the recipient is not known and no IP exists to propagate the call furthermore.
If the recipient domain is a domain name, the SRV DNS search is done, then if failure to get a SIP server IP, a A DNS search.
The DNS SRV (then DNS A) search is the "last chance" search.

The NAPTR search for E164 phone numbers (also called "ENUM" mode) enters in the global routing scheme of the S5000.
E164 database searches are to be prepared through StaticEntities "OUT" and enters within an "Embedded Services" routing plan that can mix several searches with backups: search thru e164.arpa then within freenum.org then myprivatenum.com, etc.

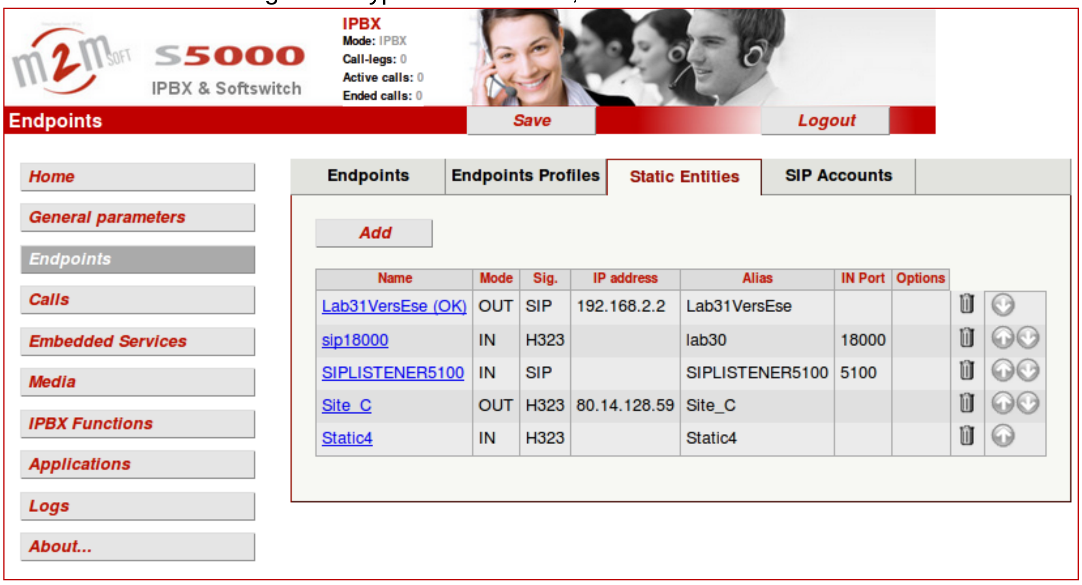


Fig. StaticEntities with DNS ENUM entries PRV1 and PRV2

The IP address for the StaticEntity OUT must be set as follow:
sip:enum:<base suffix> (example: sip :enum :e164.arpa)
(see Static Entity parametering chapter)

6.8. Resilient solution

A S5000 solution can be resilient from two **standards mechanisms**:

- alternate gatekeeper mode (H323)
- automatic discovery of H323 gatekeeper/SIP Call Agent

NOTE:
A specific very high capacity S5000 design is available on request (option).

6.8.1. Alternate Gatekeeper

The S5000 allows for advertising of an alternate Gatekeeper (a second S5000 for example) in reply to endpoints registrations.
With this feature, if supported by the terminal, the endpoint will automatically redirect its signalling towards the alternate Gatekeeper as soon as it detects a failure on the main S5000.

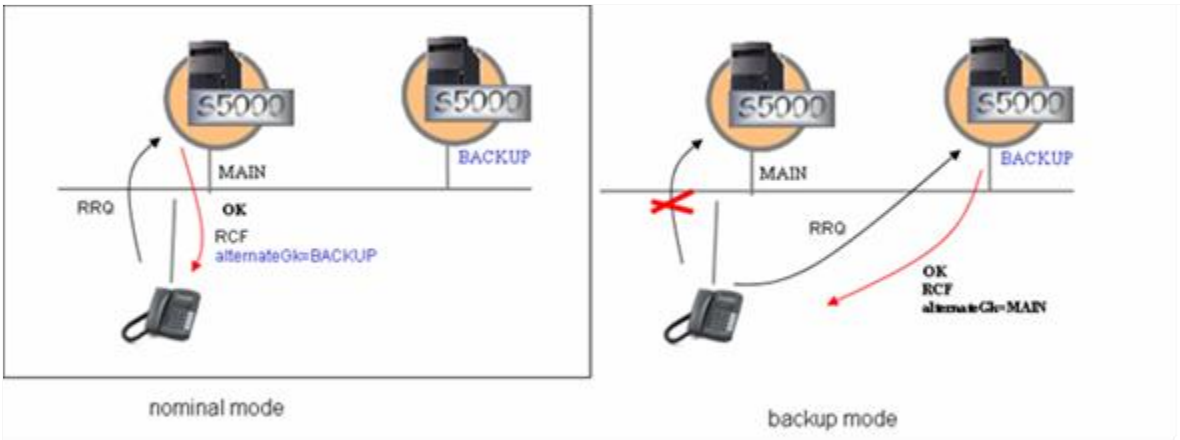


Fig.46 Alternate Gatekeeper design

6.8.2. Automatic discovery

In this mode, the endpoints have the ability to send their registration requests in multicast mode.
H323/GRQ messages are advertised amongst the local network.
SIP/REGISTER messages are advertised amongst the local network.

If multiples S5000 are active, the one that have the multicast flag on (-t multicast start option set) will accept the endpoint registration.
When all endpoints are registered, a second/backup S5000 can be started with this very flag multicast set. In case of failure of the first server, all endpoints will automatically re-register with the new S5000.
This process is a transparent mechanism from the user point of view.

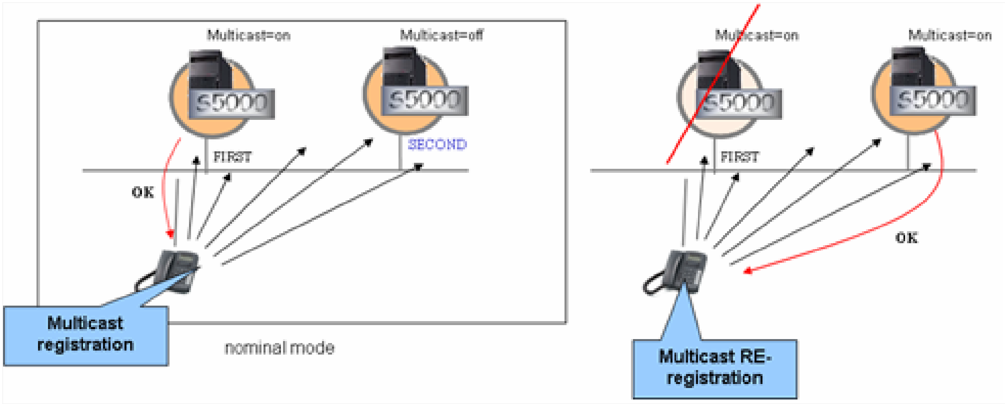


Fig.47 Automatic discovery design

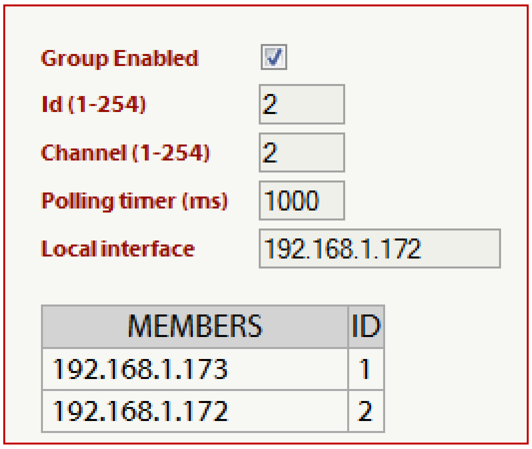
6.8.3. S5000 Groups

This feature is under license option.
The S5000 Groups is an automatic resilient mechanism based on the automatic discovery feature of H323 and SIP.
The Groups mechanism can work with or without Cluster and Database options (see farther).
The S5000 Groups mechanism allows for:

- automatically secure a gatekeeper/call agent with one or more backups
- automatically add new backups servers
- automatically swap to backup servers then to nominal server
- have all servers started at the same time

The S5000 Groups contains any number of S5000.
Any number of Groups can be active at the same time, on the same network.

Each S5000 in a group automatically discovers its adjacent ones.
It works on a master/slave mode. Every S5000 in a group as a different id, the highest id defines the master S5000.
Only the master registers the endpoints.
Slaves only wait to be master.
At any time, a slave can become the new master. A new master, in case of failure of the current master, is elected amongst the slaves, from the highest id left.
All the process is dynamic and every S5000 in a group permanently listens to each other's.



Tasks:
Configure Groups parameters as described in General Parameters § 5.3.5.

- Set an Id for the S5000 member.
- Set the channel as group Id.
- Define the timer polling.
- Select interface by IP for listen other members.

Automatically, every S5000 discover its neighbours.

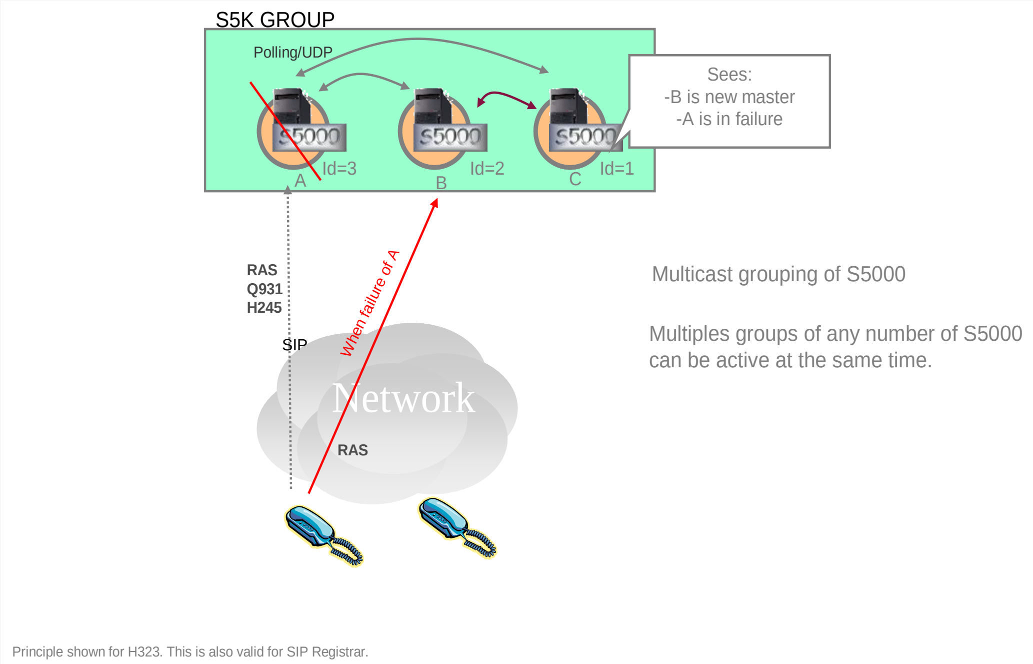
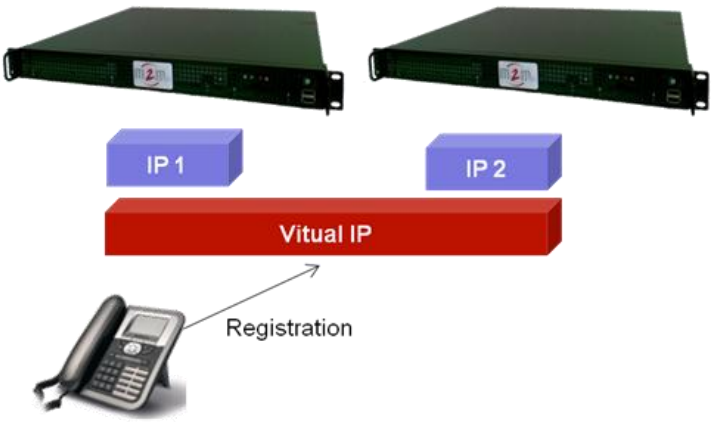


Fig.48 Group of S5000 resilience design

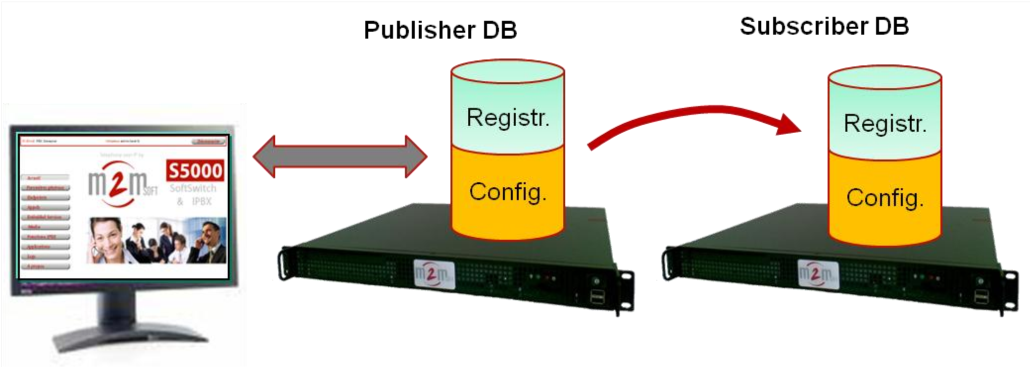
Groups - Cluster Option :

This option allows to set a virtual IP address for the group to be dynamically assigned to the S5000 elected master. Thus the endpoints will register using this virtual IP regardless of the active server.
Each s5000 cluster member has
- a first Ethernet interface (eth0) used for VoIP (SIP/H323) protocol exchanges with:
  - a physical IP address (ex: 192.168.0.172 / 24)
  - a virtual IP address in the same subnet that physical one (ex: 192.168.0.174 / 24)
- a second Ethernet interface (eth1) used for groups polling control and license with a physical IP address in another subnet (ex: 192.168.1.172 / 24).



Groups - Database Option:

This option allows to manage a unique configuration for the cluster (instead 1 configuration per each s5000).
The primary node (initially master) works as a "Publisher" database, the secondary nodes run as "Subscriber" databases. Only publisher node replicates the configuration to the publisher nodes.
The database option allows also to replicate registered endpoints. Thus when the primary node goes down, the next secondary node became master get immediately the endpoints contexts to be able to route call to them.



6.8.4. Automatic restart with jWatchdog

The S5000 can be delivered with a companion product named JWatchdog.
JWatchdog monitors the S5000 program periodically. When S5000 has a failure and does not run anymore, JWatchdog restarts it to ensure non-stop service
JWatchdog can be found in the <add-on> directory of your official installation CD-ROM.

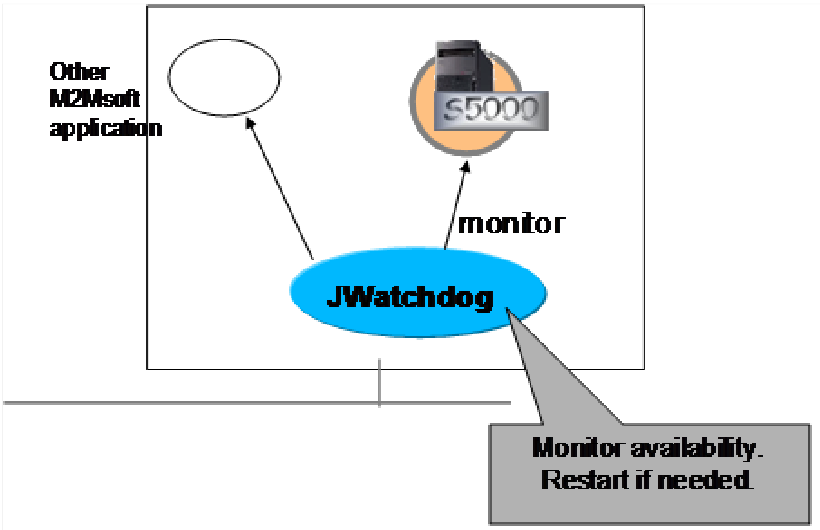


Fig.49 Watchdog design

JWatchdog can monitor multiple M2MSoft programs, products and even customer handmade applications based on M2MSoft API.

JWatchdog is to be installed on the same host than your products and applications to be monitored.

JWatchdog needs a Java Virtual Machine runner to execute.
All parametering are done within a jwdog.ini file.

How to start JWatchdog

jWatchdog is delivered with:

- jwdog1.0.jar
- jwdogstart.sh, Linux starter
- jwdogstart.bat, Microsoft Windows started
- wdog.ini sample

Customize your starter script with CLASSPATH parameter and java runtime path.

Configure your wdog.ini file with:

- what application you want to monitor
- what is your scanning period per application
- what is the startup command line per application

wdog.ini file is built as follow:

Block	Parameters	Description
[Application] (multiple)		
	name	Name (user free string) of the application to monitor. Example: name=s5000
	args	Startup line to restart the application when it does not respond to network polling. Example: args=sh ./s5kstart.sh
	period	Time in second between polling Example: period=30 (stands for 30 seconds)
	port	IP address : tcp port to connect periodically. Important: jwdog monitors exclusively TCP ports on the same host. Example: port=192.168.0.9:8000

Wdog.ini example

```
[Application]
name=s5000
args=sh ./s5kstart.sh
period=10
port=192.168.0.2:8000
[Application]
name=userApp
args=sh ./startAPP.sh
period=2
port=192.168.0.2:9000
```

6.9. RSVP service

As an option the S5000 can handle standard RSVP resource reservations for the calls going through a routers cloud.

RSVP can be used in an automatic way through the StaticEntities elements (see § 5.4.5), allowing for any non-capable RSVP endpoint to benefit of RSVP through the S5000 and allowing a complete point to point reservation for RSVP capable endpoints.

To activate the RSVP, first choose a link between two S5000 and declare a static entity to reach each other.
Select RSVP mode on these StaticEntities and the different parameters necessary for the selected QoS:

- RSVP refresh period (in ms)
- RSVP Reservation Style (Fixed Filter , Wildcard Filter or Shared Explicit) (*****)
- RSVP Bucket parameters (Token size and rate and Peak data rate)
- RSVP RSpec parameters (Guaranteed rate)

The reader must have minimal information about RSVP mechanisms (RFC2205 compliant) to make a correct use of these parameters.

(*****) *Only Fixed Filter style allows for distinct resource reservation per flow, per call. Other styles "share" the resource and could not be suitable for most applications*.

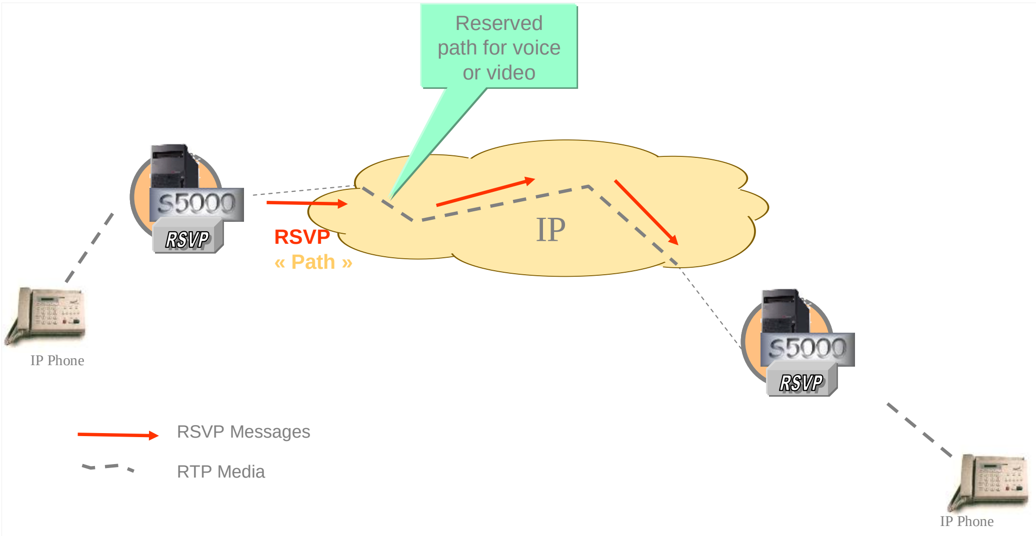


Fig.50 RSVP service design

Every call, originated or terminated within a StaticEntity with RSVP option activated will be applied RSVP messages exchanges for every voice/video flow in the call.
When only one party supports RSVP, the call is not affected by the RSVP mechanism as it cannot be set if not both parties are RSVP capable.
When both parties support RSVP, the call is automatically released whenever a reservation error occurs. At any moment in the call, .a reservation can be cleared by any router by a lack of resource of reservation pre-emption. (RSVP PathErr or ResvErr messages). This causes automatically the call to be released by the S5000.

The S5000 automatically refreshes the path according to the refresh period set. (Time Values parameter)

6.10. *Secured calls with Transport Layer Security*

NOTE: this chapter is not a cryptography manual and the interested reader must access to the rich information available on internet to emphasize its knowledge about X509, asymmetric and symmetric keys.

The S5000 embeds a TLS (Transport Layer Security) layer compliant with TLS1.0 (RFC 2243).
TLS applies on TCP links and establish a handshake and challenge between parties before the exchange of signed and encrypted application data can be done.

TLS is optionally used:
- **for VoIP calls**
- **for secured HTTP (HTTPS) access to the system.**

6.10.1. Certificate and private key needed

In order for the encrypted connections to be done, the S5000 will send a certificate to its parties.
This certificate is an X509v3 compliant file (asn1 format with .DER extension) that contains the S5000 server public key. This will be used by the client to crypt his data to the S5000.
In order to process the handshake with its parties the S5000 needs also a private key file in PKCS#8 format. This will be used by the S5000 to decrypt the received data.

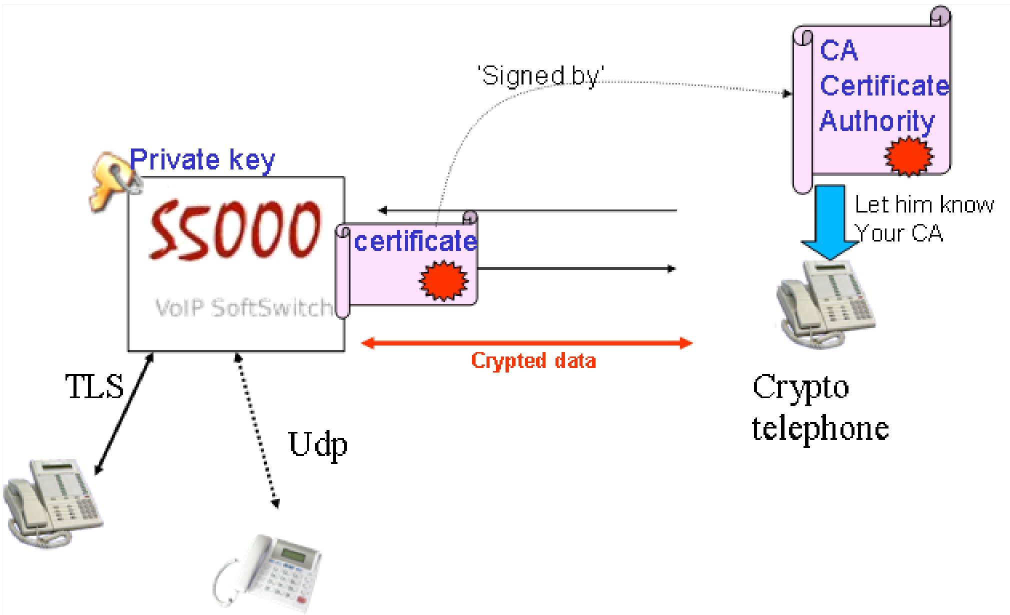


Fig.51 S5000 TLS principle overview

The certificate and the private key are the two PKI elements needed to parameter the TLS layer within the S5000.

Certificates and keys files must be installed in: <installation_directory>/cert

6.10.2. Configure Security parameters

The General Parameters (§ 5.3.7) enables the security parameters to be set.
These are:

- The S5000 certificate: you may obtain/generate a X509v3 asn1 DER certificate file.
- The s5000 private key file, in PKCS#8 format, asn1 DER.

Certificates and keys files must be installed in: <installation_directory>/cert

The S5000 certificate must be signed by a CA (Trusted Certificate Authority) trusted by the TLS client. Several companies (Verisign, etc.) offer CA signed certificates and these are trusted by default within the TLS client phones or internet browsers, but you may simply start with a self-signed certificate. This implies that you generate:

- a CA certificate (you are your own trusted CA) ; this one may be installed once on your TLS clients phones for them to accept any S5000 certificates
- a S5000 certificate signed from the previous CA
- a S5000 private key

NOTE: openssl (http://www.openssl.org/) is one tool to generate a self-signed certificate and private key files.

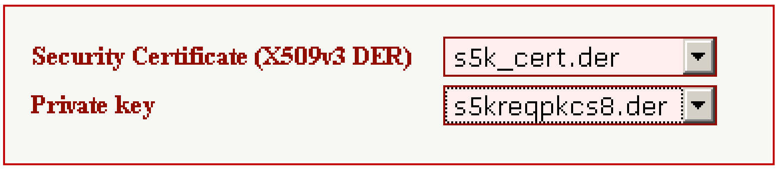
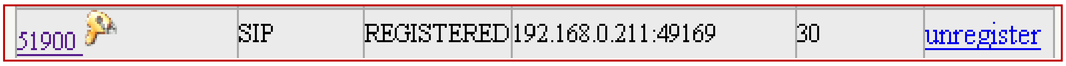


Fig.52 S5000 TLS keys files parametering

Once an endpoint has registered with TLS, it appears with a small key icon as shown below (in Endpoints page):



TLS calls are performed on TCP packets and content is totally opaque to eyedroppers.

a) Auto Generate your S5000 certificates files with openssl and HTTPS access

The following tutorial is based on the openssl version 0.9.8g. The reader may adapt this tutorial to its own openssl version.
NOTE: you may generate the following files on another system that the S5000 platform.

1. Defines/creates your CA Certificate Authority. This will be YOUR ADMINistrator certificate and will be necessary to create all other certificates for any web sites.
   1. Creates a directory
      1. `mkdir myCA`
      2. `cd myCA`
   2. Generate an RSA private key of 1024 bits length
      1. `openssl genrsa -out` **`myca.key`** `1024`
   3. Generate a CA certificate (Certificate Signing) self signed, for 5 years (aka 1825 days)
      1. `openssl req -new -x509 -days 1825 -key` **`myca.key`** `-out` **`myca.crt`**
      2. `Answer: (example)`
         1. `Country Name : FR`
         2. `State or Province Name : MidiPyrenees`
         3. `Locality Name (City): Montrabe`
         4. `Organization Name : M2MSOFT`
         5. `Organizational Unit: Research`
         6. `Common Name (your name): Bosqued`
         7. `Email :` support@m2msoft.com

         `myca.crt (X509 format) file is built.`

2. Create your server certificate
   1. Generate an RSA private key of 1024 bits length
      1. `openssl genrsa -out` **`mys5k.key`** `1024`
   2. Generate a certificate (Certificate Signing Request) for the S5000 site for 2 years. (aka 730 days) . Produces a .csr.
      1. `openssl req -new -key` **`mys5k.key`** `-out` **`mys5k.csr`**
   3. Sign the certificate with your CA key. DER binary format file is needed at end.
      1. `openssl x509 -req -days 730 -in` **`mys5k.csr`** `-CA` **`myca.crt`** `-CAkey` **`myca.key`** `-set_serial 01 -outform DER -out` ***`s5k_cert.der`***

      When/if prompted for password: just press enter (leave blank).
   4. This private key file is not in PKCS#N format.
      Generate a PKCS#8 format private key file (for the S5000), as DER binary file.

      `openssl pkcs8 -topk8 -nocrypt -outform DER -in mys5k.key -out` ***`s5kreqpkcs8.der`***

Now, just load the s5k_cert.DER and s5kreqpkcs8.DER files into the S5000. Then **connect through HTTPS**.

6.11. T120 Proxification

This feature is mainly used within H323 architectures.
Media proxification is often needed as an IP masquerade for external servers. Data proxification meets the same needs and T120 is a widely used recommendation (last revision from jan 2007) for data communication on multimedia conferences (ITU –T 121 to 127).
Microsoft Netmeeting is the major reference implementation.
The reader may refer to these recommendations for further interest.

Voice and video proxification is achieved through the MTP (Media Termination Point) module. T120 proxification is done through the S5000 itself.
T120 proxification is selected within the General parameters tab of the S5000 web interface and is enabled or disabled for all T120 flows. (See General Parameters chapter and **T120 enabled** field)

S5000 acts upon dynamic detection and analysis of H245 capabilities and actual T120 requests to offer a multi channels (TCP based) T120 relay.

NOTE: The TCP ports pool is used, as specified within General Parameters tab.

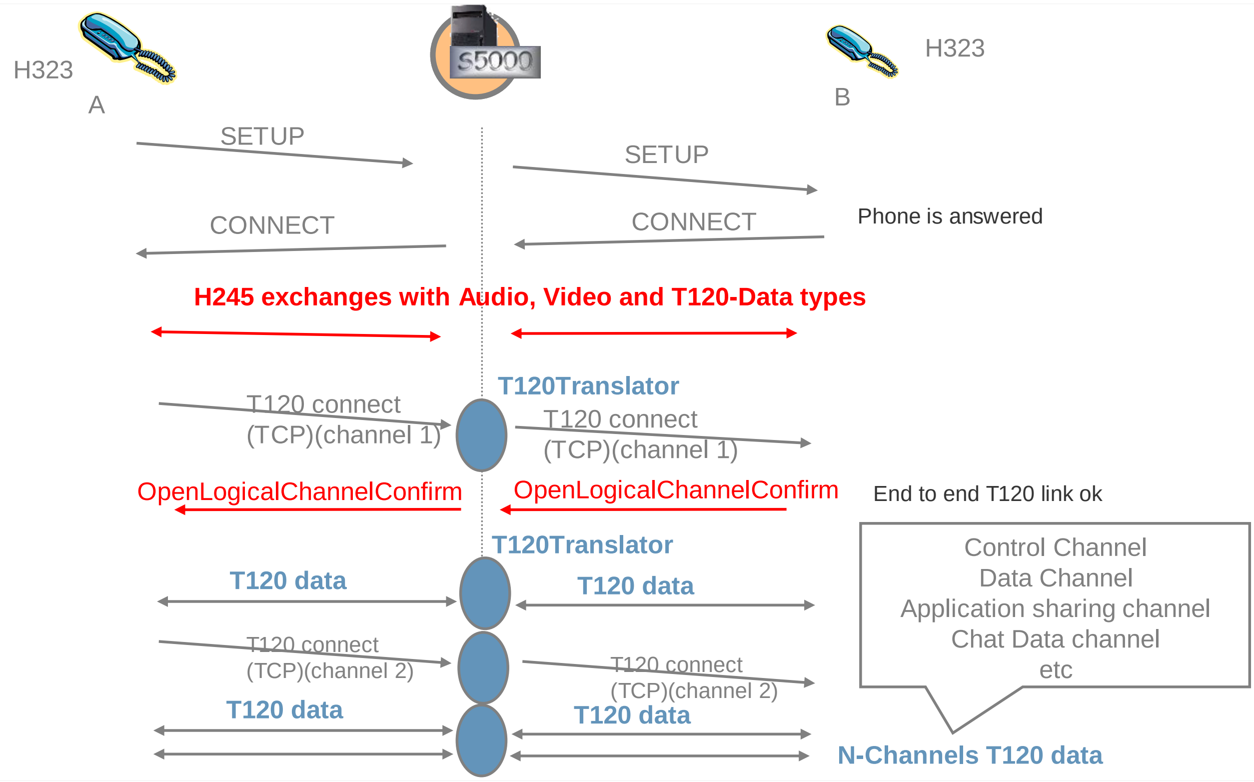


Fig.53 T120 channels proxification view.

6.12. IPBX mode

The S5000 embeds enterprise telephony functions, most of what is expected to be found on plain old PABX systems. The S5000 must be set into **iPBX** mode (see General Parameters) for the following.

IPBX Features

These features include:

- **internal and external calls**
- **internal calls on short numbers**
- **call forwards**
- **call transfers**
- **call pickup and intercept**
- **3 parties conferences**
- **call filtering ("patron/secretaire")**
- **multiples lines handling** (up to 9 per basic telephone, and much more on advanced phones)
- **call hold/retrieve**
- **call select**
- **lines supervision** on selected switchboard phones **(BLF Busy Lamp Field feature)**
- **management of external Ip Phones through internet line, even located within private networks**
- **special "forward" conditions ("closed offices", etc.) can be shown on BLF (Busy Lamp Field) of IP telephone**
- **Call Waiting Indicator (as a "beep" sound while in a call)**
- Etc.

The S5000 in iPBX mode can be used with PSTN Gateways (SIP or H323) to offer a complete IP-PBX system.

The S5000 Enterprise is an M2Msoft integrated HW and SW with all the enterprise features and PSTN connections.
Ask our sales representatives for information.

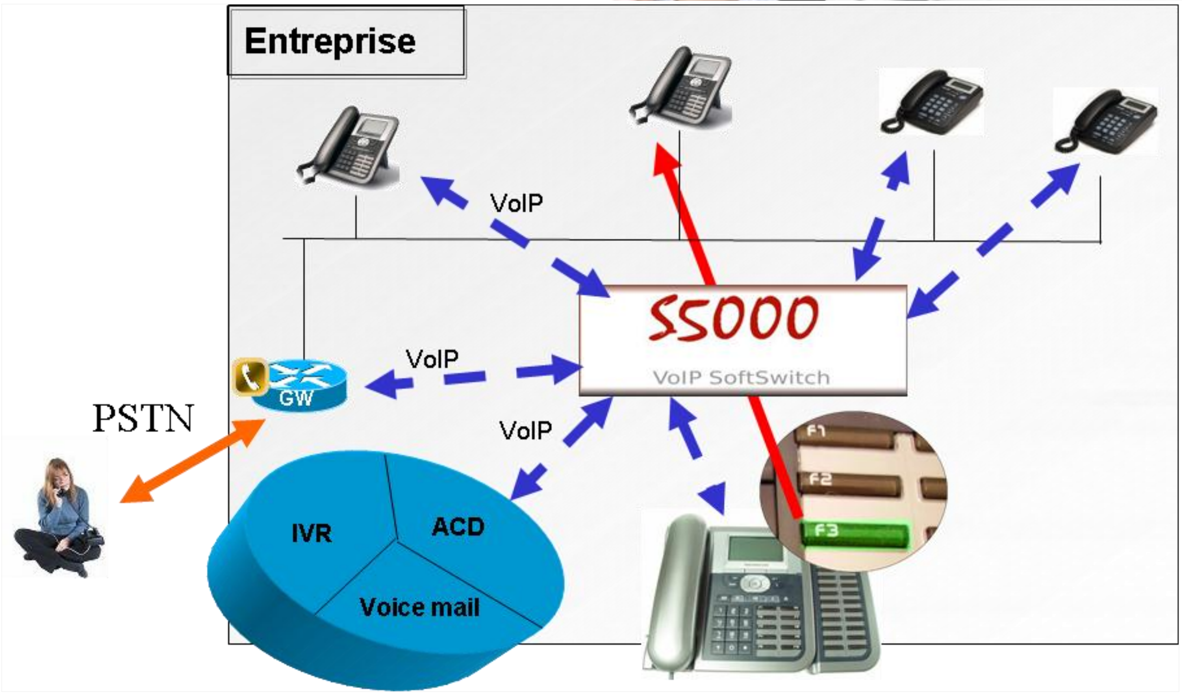


Fig.54 S5000 iPBX within the enterprise

The above figure depicts a typical S5000 with iPBX functions use case.
The company simply needs some IP Phones, a PSTN gateway and an S5000 to handle its telephony for employees and customers calls.

The iPBX services are available on IP Phones that handle:

- DTMF using RFC2833 or SIP INFO or H323 UII (for Virtual lines feature)
- (optionally) Hold/Retrieve and Xfer through SIP ReInvite and REFER requests

Supervision line and call intercept

On capable phones, the S5000 iPBX can monitor users/phones status and pilot phone led (Busy Lamp).
This can be useful for a switchboard position to look at people busy states as well as for call intercept:

- simply pressing the corresponding light on (Busy Lamp) performs call intercept on the ringing call on the ringing phone

Calls transfers and multi lines

Pick up a call, put it on hold, then dial a new call to B, then transfer the call with some simple dtmf sequence or simply hang up: as soon as B pick the call, it is transferred.
Situation as simple as that one are naturally managed within the S5000 and ease the enterprise telephony deployment.
Any number of concurrent calls can virtually being held from a single phone.
Simply go and tabulate thru the calls:

- either from your phone function keys (advanced phones)
- either by pressing #1(call one), #2 (call two), etc.

Announcements

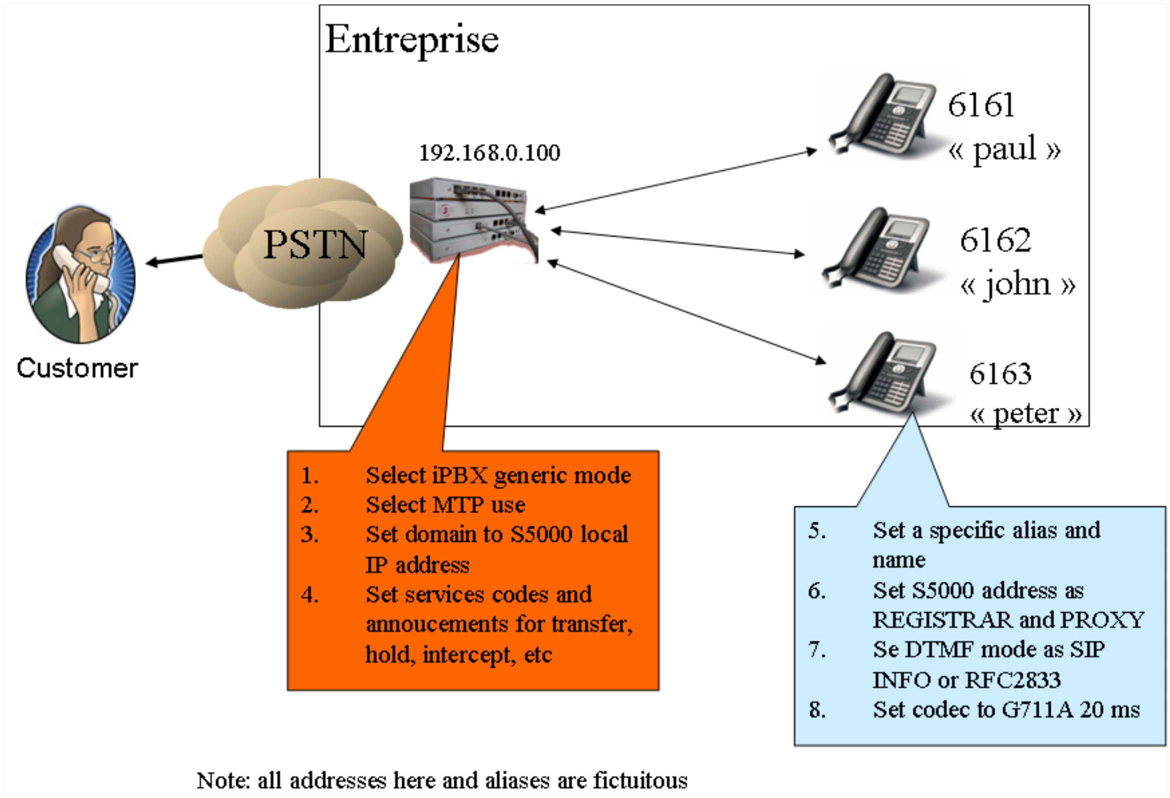
Several voice/music prompts are used in iPBX mode.
Music on hold (server generated), pre-connect announcement, etc. and are fully customizable (see Recorder function chapter) thru G711 files.

Closed offices management and PBX Rules

A number of redirect rules can be set within dates, and hours or simply activated/deactivated on demand.
This can be used to advertise a closed office message for example, on evenings and week ends.
This can also be used to transfer a complete switchboard to an alternate location for example.
Every rule can be activated/deactivated using an internal IVR.

Tasks to set up an S5000 Enterprise environment

We assume here you have your S5000 installed and set up onto your company LAN (i.e. IP fixed private address for example).



1/ Be sure to have "iPBX generic" mode activated.
Go into General Parameter page and check "iPBX generic" checkbox is checked.

2/ Be sure your SIP domain is set to the actual IP address of the S5000 (eiher private here or public IP if S5000 is directly in internet)
Go to General Parameter page, and
check SIP view, with SIP domain
check SIP UDP listener at least is selected

3/ Be sure you are working with MTP
Go into Media page and check Media Termination Point to see an MTP server set and a light on for an MTP server.
If not, please declare a new MTP and check you have '*mtp*' application running on your host or any remote host.
Go to MTP Rules and check there is a rule for all calls.

4/ Define and set the different services codes: hold, transfer, intercept, etc.
Go into iPBX Functions page and set the codes within the page.
(see IPBX page chapter)

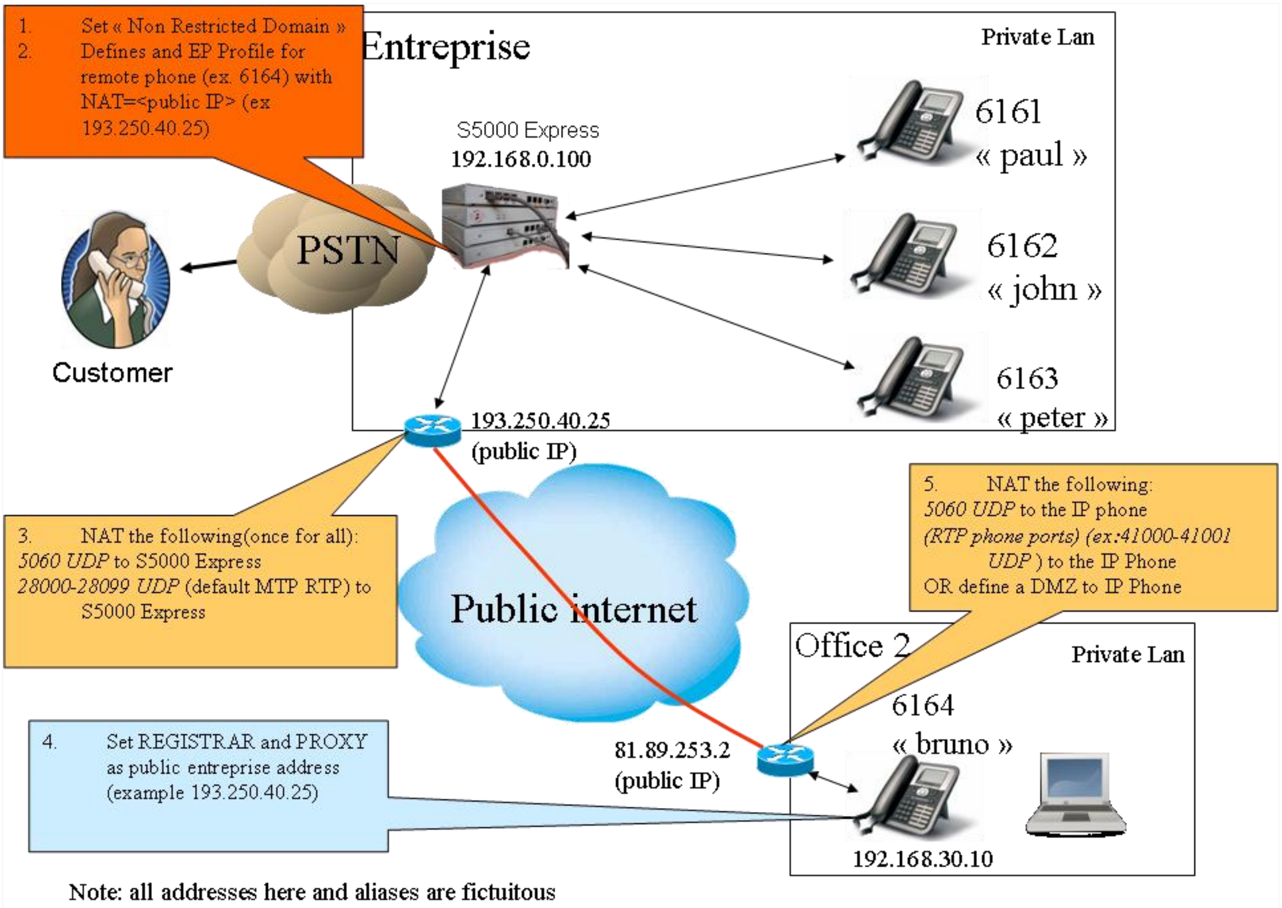
5 to 8/ It is time to configure your local phones if any
Enter them an alias and user name, the IP Address of the S5000 as REGISTRAR and SIP Proxy, with port 5060.
Select the IP Phone into DTMF Mode SIP INFO or RFC2833.
Select the IP Phone codec as G711A, 20 ms preferred packet size
How to configure this depends on your phones.
Virtually all SIP with DTMF capable phone are suitable but the S5000 Enterprise is fully validated with THOMSON ST2030 IP Phones. Special parameterization documentation is available for these phones upon request.

Check within S5000 Endpoint page that you “see” all your phones: you are up and running.

If you have external IP Phones on remote private or public site, it is all within the next chapter.

Tasks to set up remote private IP Phone to enterprise S5000 Enterprise environment

We assume here you have your S5000 installed within your offices and set up onto your company LAN (i.e. IP fixed private address for example). Your offices are connected to public internet thru a router and a public address.



To start, the S5000 domain as seen from the remote phone will not be the same as the one shown within the local phones: your S5000 is here running a private domain (i.e. its local private IP address). Then the external phones will need some NATting to dialog with the local company phones.
Let’s see all this.

1/ Allow S5000 to accept other domains than itself
Go into General Parameter/SIP page and check “No Strict Domain Control” checkbox is checked. This allows external phone REGISTER request that contains URI with offices public addresses to be accepted.

2/ Defines an EP profile for the new external phone with NAT
Go to Endpoints/Endpoints Profile page, and
Create or modify if it exists, the new phone profile
Add a NAT field with the public address (as shown from the remote IP Phone)
This will force all SIP requests coming from the S5000 to this remote phone to have rewritten addresses on some headers and fields (SDP, Contact, top VIA, From, To, particularly)

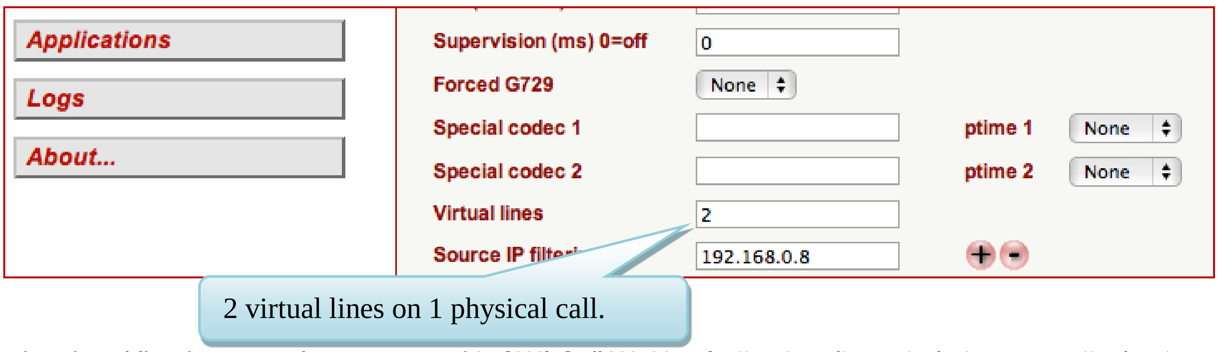
3/ NAT ports into the enterprise router
It is necessary to route external ports here such as:
5060 UDP: this will be the main entrance for the REGISTRATION of remote endpoints.
28000-28099 UDP: these are the default UDP RTP range as advertised by the MTP to get the media flow from the external phones to go to MTP

4/ Within the remote Phone, set it to register to the Enterprise S5000
Define the REGISTRAR and PROXY with the public Enterprise address

5/ NAT ports into the remote site router
It is necessary to route some ports here such as:
5060 UDP to the IP Phone (in case several IP Phone are used, use different ports here, 5070, 5080, etc.)
<ip phone media port> UDP: the RTP port as advertised by the IP Phone must be directed to the Phone. (consult the telephone manual for this) It can usually be fixed within the phone.

6.12.1. Virtual lines: 1 call, N lines

The S5000 can use a unique feature to handle multiple communication lines through a low bandwidth network or one line limited endpoints.
This mode is dedicated to work in IPBX mode. Define a set of endpoints to have a limited number of lines: this enable the virtual lines mode for the endpoints (this is defined through the StaticEntity listener that will force all registered endpoints through him to get the feature). This endpoint will receive only one physical call and will be able to manager several calls within this single physical link by use exclusively of DTMF codes.

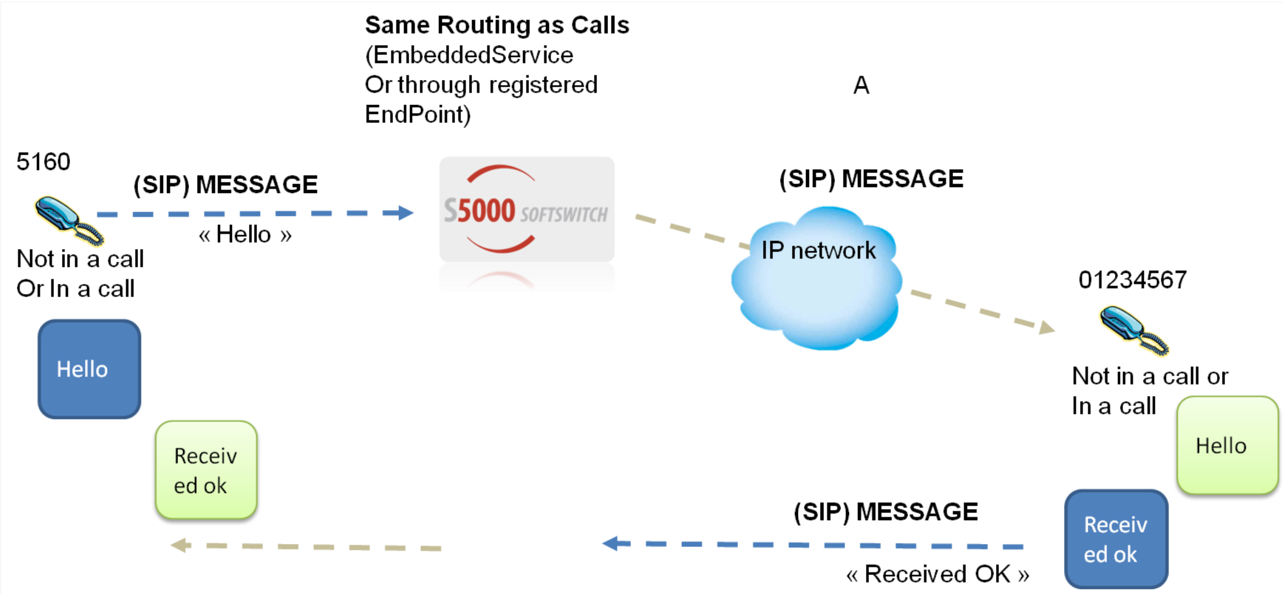


The virtual line is a complete system with CWI Call Waiting Indication, line tabulation, auto display ‘on screen’ updates (after a line tabulation of transfer, one needs to “se” the remote party name/number), error messages (when the maximum number of lines is reached), conferences. The application can take complete control over the system and set/reset special SIP parameters:

- Displays (from/to)
- Priority
- GenericParams
- Accept/reject hold
- Accept/reject conference creation
- Accept/reject conference addition
- Accept/reject transfer

6.13. *Short Message Service*

The S5000 handles the SMS as defined by RFC3428 (SIP).
SMS can be issued from a SIP trunk, a registered endpoint, a non-registered entity. The same routing process is performed as defined for the calls: embeddedServices, registered table.
An SMS can be routed and transmitted from and to endpoints in a call or out of a call.
The routing table is used when the call is not yet established. Once the call is established, the SMS is transmitted on the same path as the original call set up.



7. Application Programming Interfaces

This chapter introduces first the M2M-S5000 API concepts, and then API commands and events references for each of Java and C languages.

7.1. S5000 API concepts

Although the M2M-S5000 has been designed in order to enable a large range of functions directly from a set of configurable « embedded services », real world complex applications and specially user applications may possibly not be directly rendered.
M2M-S5000 has its core functions extended to one or more user external applications: Applications that can even run on a bunch of other platforms.

How to only allow a set of phones to place calls?
How to I dynamically route calls from database rules?
How to design specific signaling and media gateways?
The GKXAPI (Application Programming Interface) is the way to do so.
The GKXAPI is a set of libraries that users use to develop new code that takes control of calls, routes or terminates calls.
The GKXAPI is connected to the M2M-S5000 product.

S5000 API is called GKXAPI and it exists on two flavors:

- JAVA API: the JGKXAPI
- C/C++ API: the GKXAPI

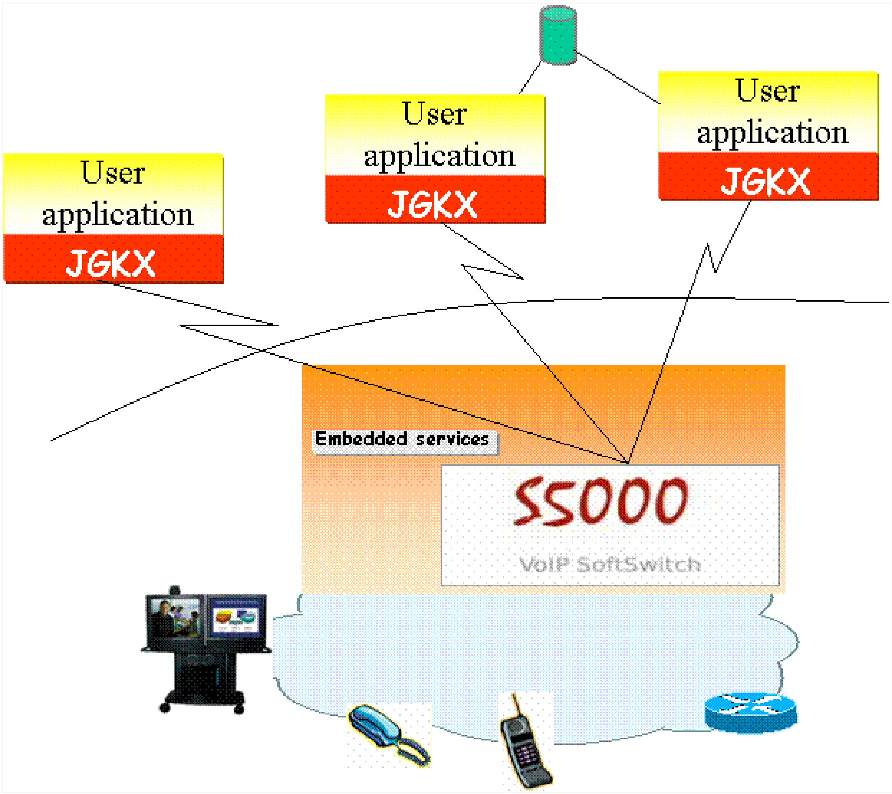


Fig.55 S5000, an IP services application server

The previous figure shows a typical use of GKXAPI with a set of applications that handle calls from various sources: terminals, video terminals, gateways, other gatekeepers, call agents, etc…
All applications are connected by means if TCP-IP with the M2M-S5000.

Routing to applications benefits of the S5000 embedded services features.
The embedded services table manages the call routing to selected applications or anywhere else based on these rules.
EmbeddedServices rules are processed in order and a stopped (not connected) application leads to the next matching rule application.

Here are the main features offered by GKXAPI / JGKXAPI:

- TCP-IP connection (same host or remote)
- Distributed applications allowed
- Any number of application connected to a M2M-S5000
- multi-threaded application: handle multiple sessions/slots at a time with a single-binary
- Multi-platform Java API
- Requests/Response and Indications protocol
- Ability to send unsolicited commands
- Endpoints Events handling : (protocol agnostic H323 or SIP) RRQ, ARQ, SETUP, ALERTING, CONNECT, RELEASE and H245 OLC/OLCAck, RSVP, etc.
- Routing based on embedded services
- Routing of events and calls can be done externally, independently of the application
- Graphical monitoring of call connected per application, per session/slot
- GKXAPI / JGKXAPI is an asynchronous API

7.1.1. Use cases: Control and Interfere call processing

One purpose, may be the main purpose of the API, is to interfere on calls processing by inspecting, controlling and modifying the routing of the call or of the terminals signaling or media.
All call control and routing applications family enter this use case.
The application does not need a call termination, this is done within an external unit.
The application deals with tracking selected signaling events, rejecting some, accepting others, with or without modifications.

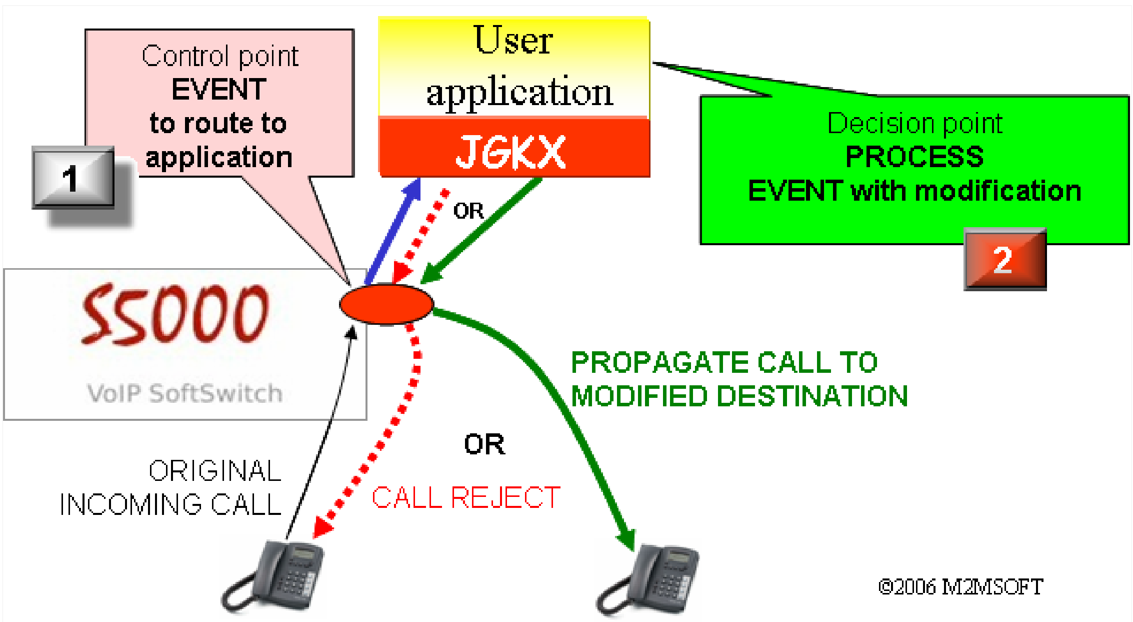


Fig.56 S5000 applications call controls principles

7.1.2. Use cases: Terminate calls with media (standard)

Direct handling of calls, call connection/termination within the S5000 and actions while the call is running, acting on media itself, DTMF control and voice services: all these needs fit within an other applications family and the **GKXAPI** provides a set of conceptual objects/entities, to connect calls with or without media, dial out calls, reconnect calls, etc.
These applications use Media Entities super object and/or SIPProxy entities and work on call legs rather than user to user calls.
The figure below shows a call directed to an internal Media Entity. The caller is connected with a file playing (G7xx codec) and the application can then disconnect the call, change the play file, connect the call to another call: join!.
The Media Entity takes care of most part of the signaling –H245 negotiation, ringing sending, automatic connection, media play-.

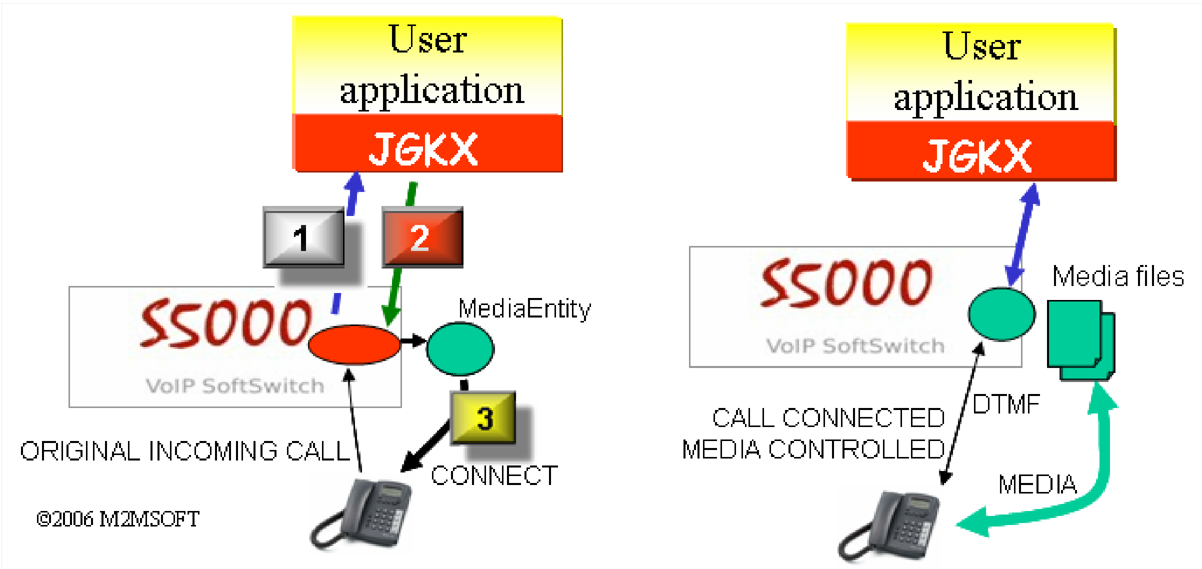


Fig.57 S5000 call termination services principles

7.1.3. Use cases: Terminate calls with specific media. Gateways design. (Advanced)

The S5000 API allows to design specific call gateways between different signaling and media data. This is achieved by giving a more advanced control on network protocol messages to the user application.
The **Generic Gateway Controller API** is a subset of the JGKXAPI and allows to send signaling messages at any time to end points. Furthermore the signaling messages can embed specific media description.
Here the user application acts as a Media Entity itself but with no predefined actions.

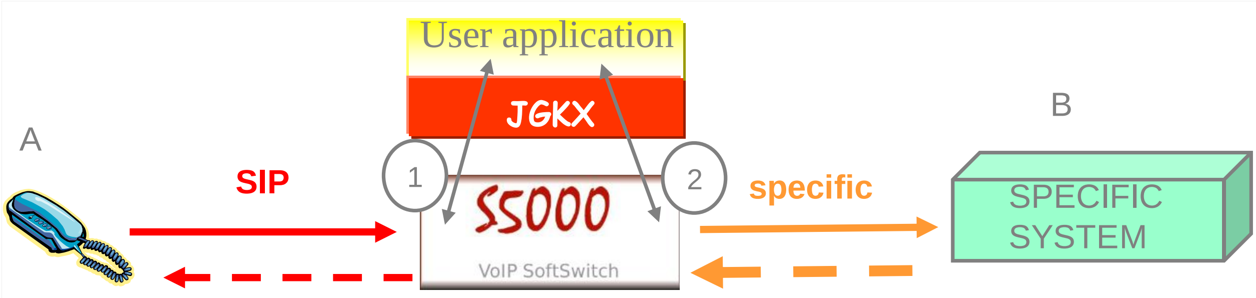


Fig.58 Generic Gateway Controler. Make different systems to dialog.

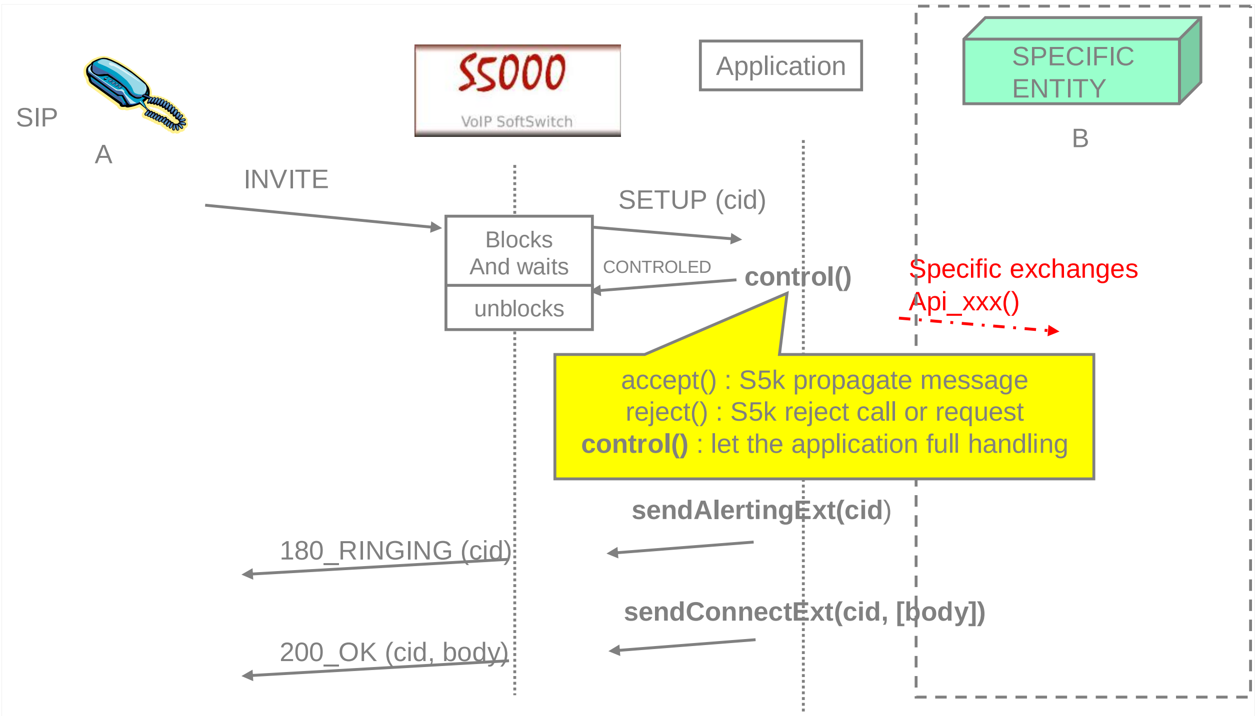


Fig.59 Generic Gateway Controller. Let the application control the signalling on each side.

7.1.4. Use cases: Transmit privateData in a call

JGKX API allows the transport of private data within a call using existing protocols fields and messages that do not harm in any way the existing system but transport important informations for applications.
Along with voice communication, proprietary data can be send using the API commands setPrivateData(), described in the API part of this document. This uses standard H323 or SIP messages.
This is done in a completely transparent way from the developer point of view.
Here is a more detailed view of the principle as given for SIP calls.

Contact SIP field is used to set this proprietary informations as stated from RFC3261 Augmented BNF for the SIP protocol:

`Contact : <sip:xx@ddd.com> ;contact-extension`

M2Msoft defines **contact-extension** as the generic parameter:

m2mpr=data;

Below is a sample of use:
Contact : <sip:897@192.168.0.34 ;**m2mpr=**786/2
Transports the private data « 786/2 ».

The figure below shows the send/receive message/actions flow thru GKXAPI:

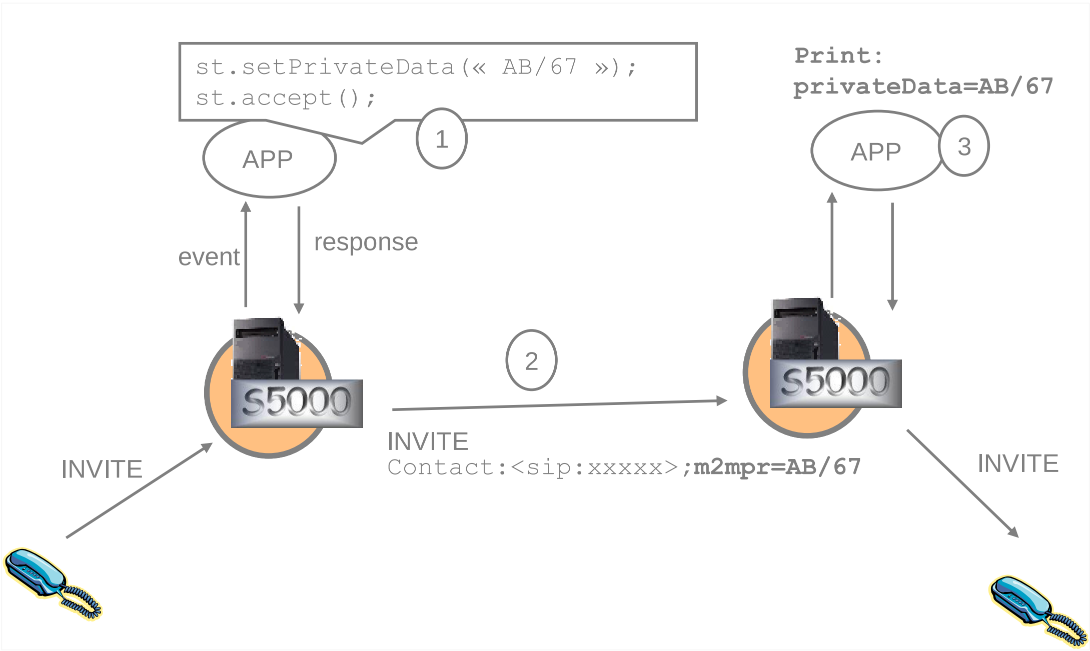


Fig.60 S5000 carries private data within call setup

7.1.5. Multithread and sessions

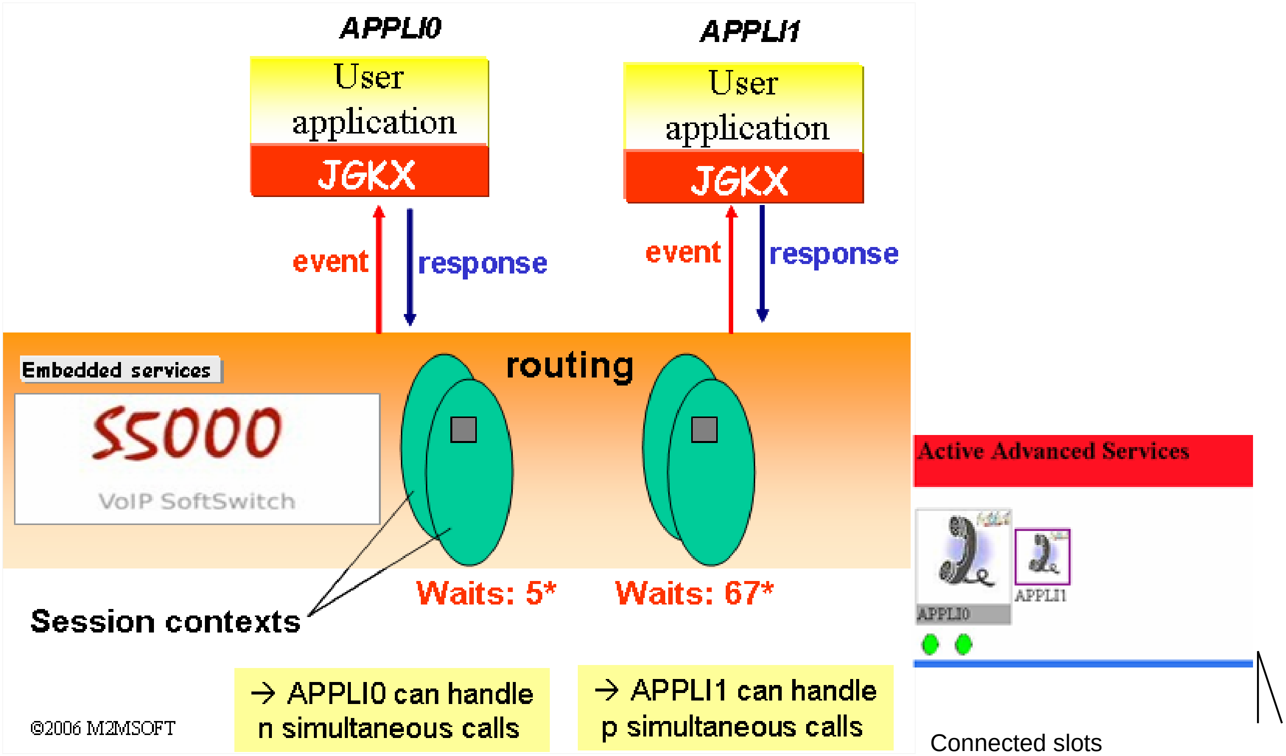


Fig.61 S5000 a generic multithreaded engine for applications

JGKXAPI allows for multi-threaded user applications, even in non-multithread environments. As shown in previous figure, for every application connected to M2M-S5000, a set of session-contexts are created according to an external setting: this is independent of the user application and allows for call capacity planning per application. The S5000 hosts its very own multithread engine for the applications, not relying upon system thread that may not exist. (when available the developer however can easily add threads to his application if needed)
Each context lives independently of the others but listen on the same route masks. Route masks are the conditions that determine an application to receive events versus another.
The S5000 embedded services defines the routes masks for the applications.

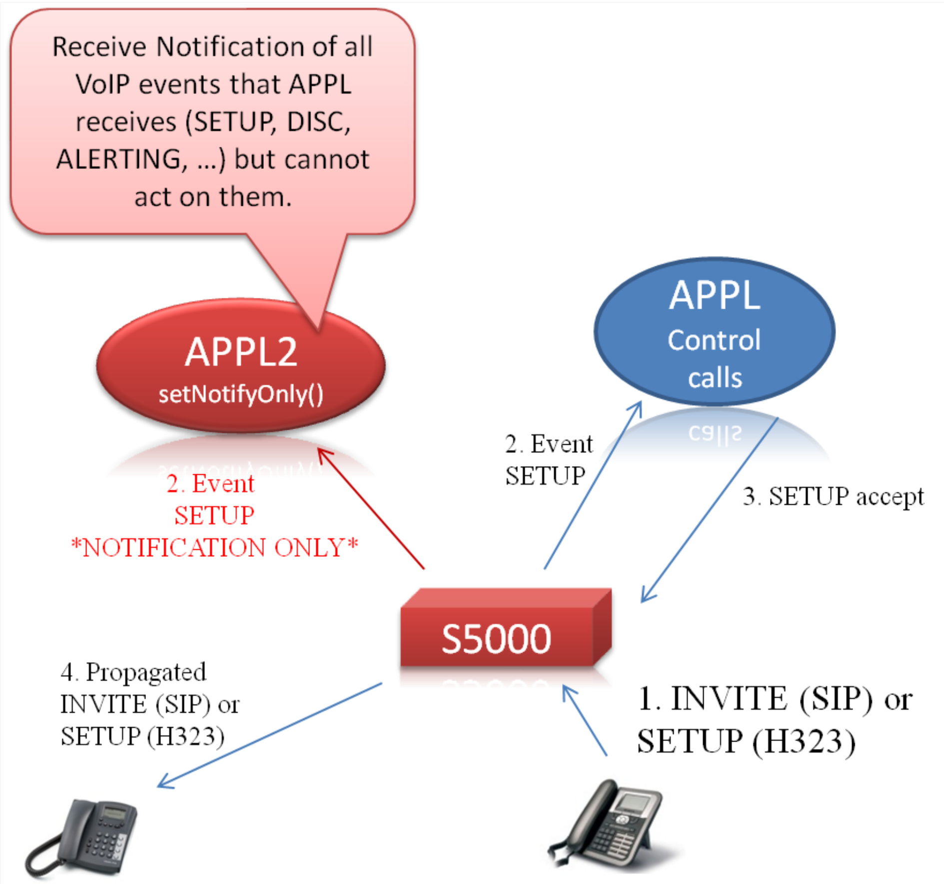
If an application is set to receive calls with called number = 5*, this application will receive events whenever a call is placed with a called number =5 and so on. All other calls will not go into this application. According to embedded services management, calls will go to another application that has waiting call contexts or directly to a new direction or endpoint.
If two applications are connected, one with routing on a mask=5*, the other with 6*, they will receive the events accordingly to their masks.
If two or more applications are connected with the very same mask, M2M-S5000 will distribute all events sequentially application per application. Once all contexts from application 1 are connected, Application 2 will receive events, and so on. As soon as connections are freed, application 1 will receive events again, etc…
When no contexts are available for a call, and if it matches an application, the call is rejected.
If an application is not waiting for any call, it is recommended to create an embedded service to direct the calls to an answering machine for example.
See § 5.6.1 and § 6.7.2 for more details about routing calls toward applications.

7.1.6. Use cases: Applications for VoIP monitoring only

One can design monitoring only applications that can work alone or with any number of other applications on top of S5000.
A simple mode can be set to have an application only receive event without the ability to interfere on them. Such applications only see incoming, outgoing calls, alerting events, releases but for all other applications.
In the figure below, APPL2 is an application that receives the VoIP events as notifications only. Of course, the event is also sent to a “real” (aka not in notifyOnly mode) active application if there is such an application and a routing within the S5000.
APPL2 can exist alone, without any other applications.

For more details, please see jgkx setNotifyOnly() method below.

NOTE: an administrative API also exist to request ‘on the fly’ information about call list, sip accounts and registered endpoints.



7.1.7. Advanced considerations

Concurrent timers

While a multithreaded application is interesting to ease the automatic management of multiple calls, not relying upon system threads it has the fall back of being sensitive to any slowdown and 'sleeps' the developer adds in his code. No more events are processed for the other connections.

> **Reminder:** The JGKXAPI engine is not relying on threads for maximum portability and thus you must avoid sleeps and large delays within your callback routines, unless you can detach threaded routines and return hand to the engine.

That is why we provide a timer system that allow to arm and manage any time waiting's within an application while not blocking the processes.
A different timer per session (i.e. per slot) can be started and managed.
On an event (SETUP for example), a timer is created (with setTimer() from the CTX object) and a moment later, a special TIMEOUT event will be generated as the timer has expired. The corresponding CTX object is given in parameter.

For example, this can be used to detect when nothing is done on a connection after a SETUP as to detect an infinite ringing …
This can be also used for periodic checks of information within databases, etc.

Session User data

The multithreaded concept of JGKXAPI applications allows for a single binary to handle multiple calls simultaneously. It is up to the designer of the application to handle his 'session context' and the events that pertain to the same call context.
To ease this management, call events belonging to the same 'slots' are attached a same "sessionId" value. The "sessionId" is a readable attribute from all event messages got from callback calls.
The developer can store values connected with the current connection (setData()) and retrieves these values later on (getData()). All these stored values are called « session context ».

For example, one can store the original called number as seen in an ARQ event. In the SETUP event, the called number can be changed and then in the RELEASE, the developer can retrieve the original called number as set in the 'session context.

Media Entities and media services

The API enables access for advanced services using the half communication concept named Media Entity. These MediaEntities live within the attached server – aka S5000 or Media Entity Server (MES) - . The Media Entity (ME) allows getting all signaling and media information within the attached server and thus acting on these at any time.
Typical applications that uses ME are:
–voice answering machine (redirect the calls to a Media Entity whenever you're not available)
–status message (to terminate a special number called)
–voice message/alert (dial out a voice message to someone phone)
–call center (let's wait incoming calls and connect incoming numbers to local operators phones as they are available)
A Media Entity s associated with an alias name (example "911" or "Player1") and a file name. (files must contain valid audio data: G711ULAW is supported, new formats are coming soon)
The alias is the way to identify and route calls to it. The file name is the voice file to play to callers or called.
Media Entities are available in H323 and SIP protocols.
An extended concept is done with Generic Gateway Controller functions that allow to move even forward in capabilities by processing nonstandard signaling and media on sip call legs.

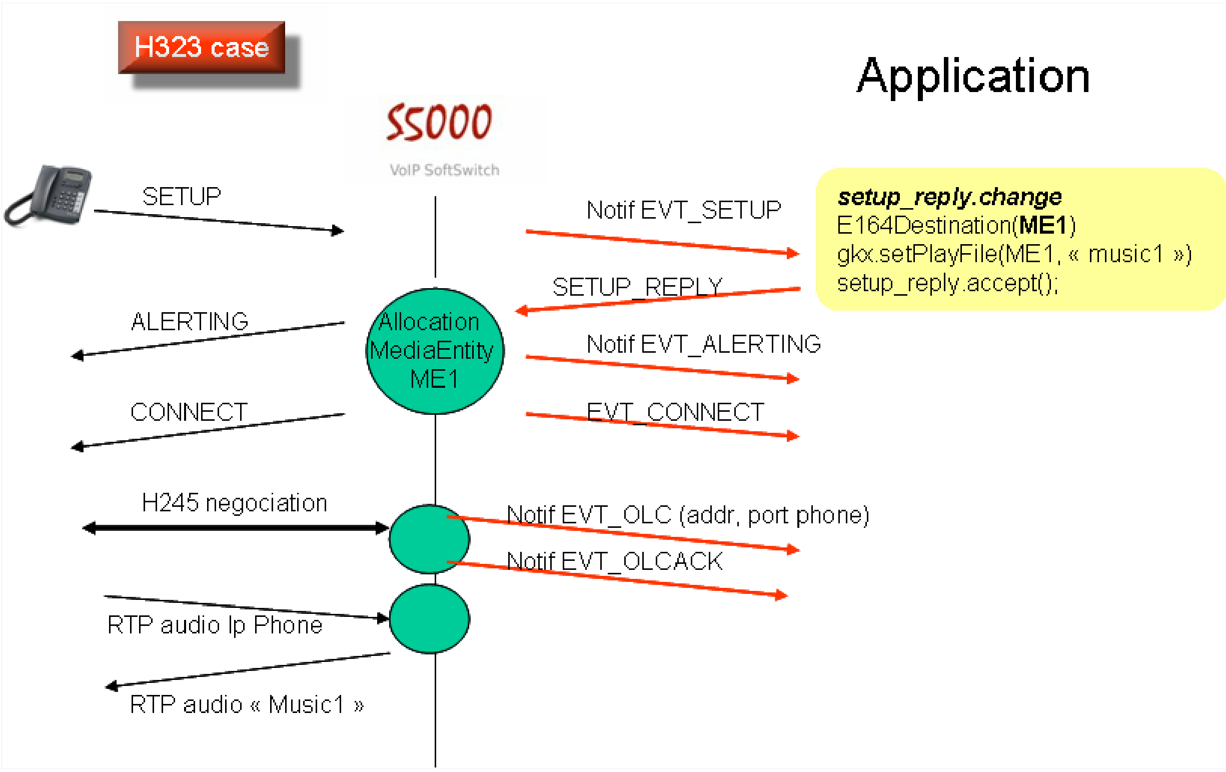


Fig.62 S5000 and mediaEntity: a simple paradigm for new services with media

Media Entity programming is simple enough to make you perform all kind of telephony services. You have to connect all your parties to Media Entities elements then join, unjoin and reconnect all of these half calls. (media codecs capabilities may be renegotiated automatically)

A sample application graph

The very next graph depicts a sample application that connects seamlessly calls through ME and perform direct voice pre-emption when the called party already in the call receives a new incoming call.
Let's say the first two parties A and B are connected to ME1 and ME2 Media Entities. A new call from C arrives for B: then we releases the A party and force B to talk with C.

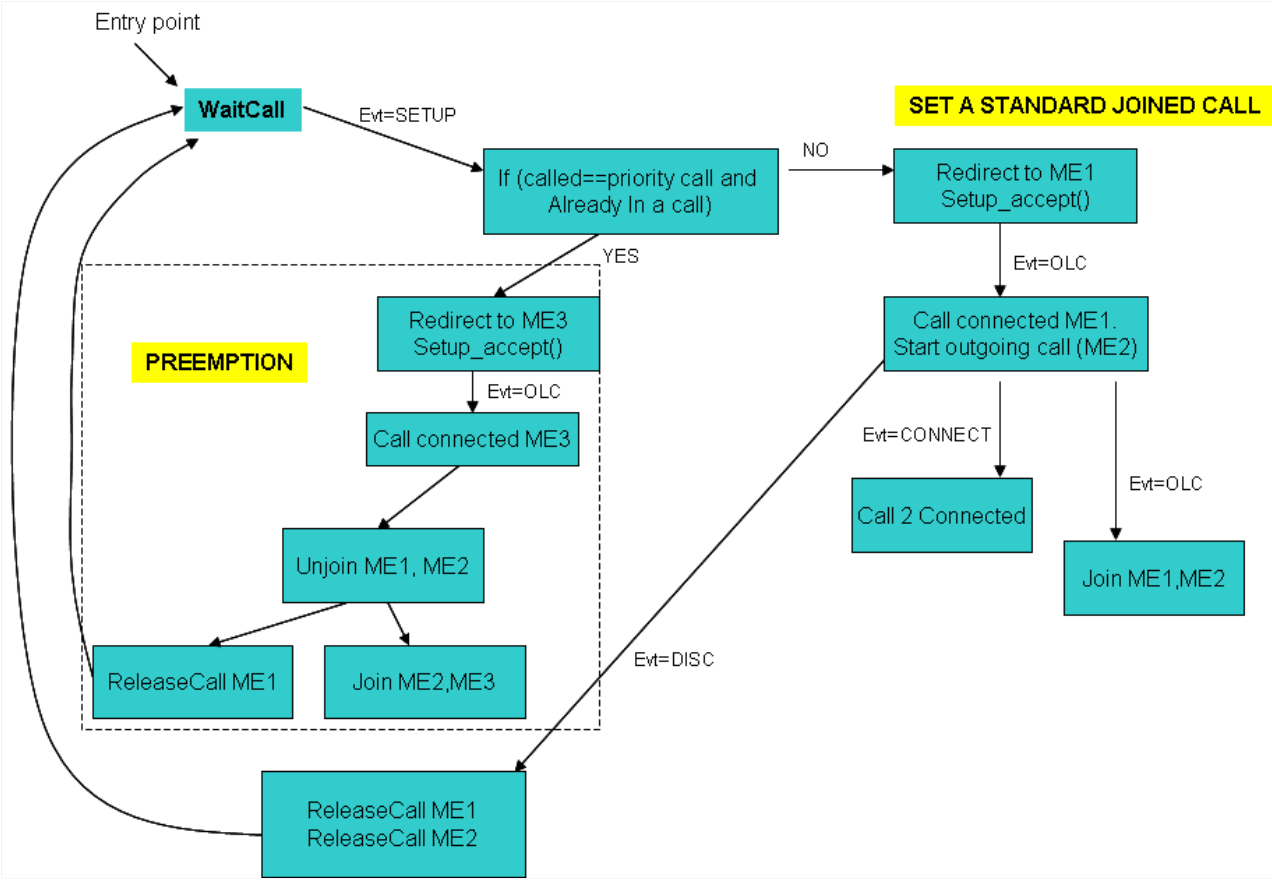
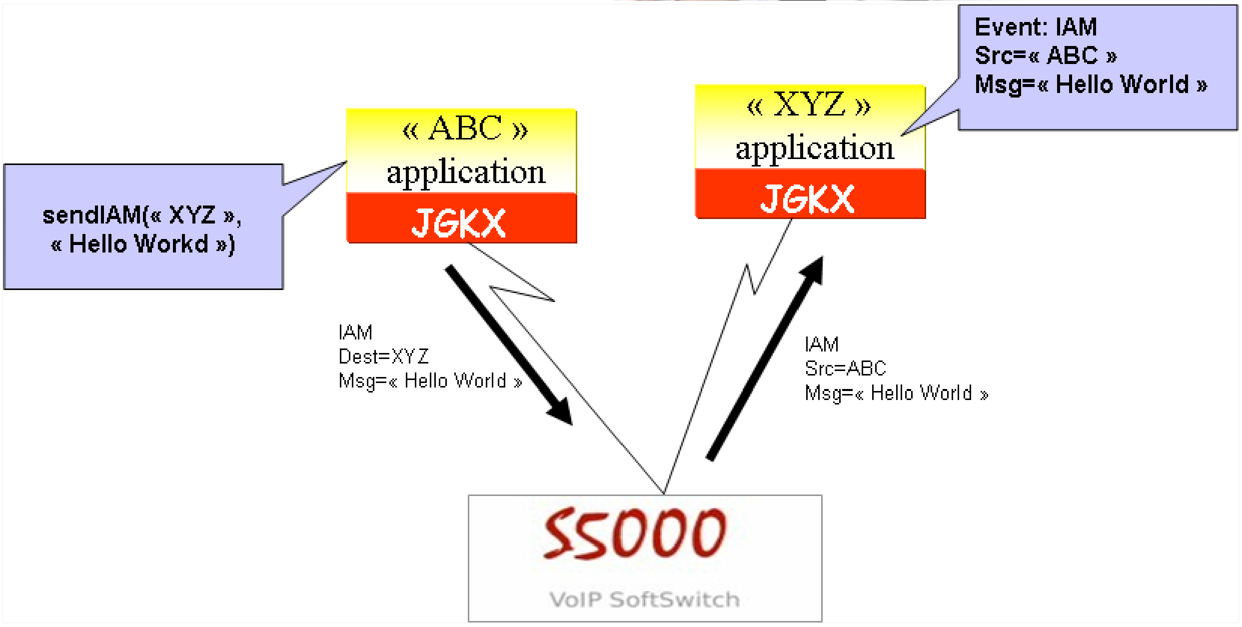


Fig.63 A sample application call flow

This document details the functions and events set associated to this MediaEntity and halve calls feature.

Inter Applications Communications (IAM)

Multiple applications can be connected to a S5000 at a time. Every application has a name (default one or user defined). Every application can send specific messages to any other applications.
This can be used as a facility to avoid implementing dedicated inter application communication by way of sockets, etc. The JGKXAPI and S5000 provide a ready to use mechanism to send/receive byte data messages between an application to another, asynchronously, thru the S5000.



7.1.8. SIP Proxy entities, Client, Server and INFO data transport

The API enables to activate SIP envelope for application by allowing them to act as a SIP UserAgent Client or SIP UA Server.
The API allows to:
- make your application to register with a sip registrar, or not
- make your application to establish connections with specific content types (currently pure private and CSTA contents)
- make your application to exchange specific, non SDP data on a connection

Each of these features can be used separately.
For SDP and embedded media functions, one must use the Media Entity.

SipProxy entities live within the S5000, as the MediaEntity and automatically handle the registration process, and the necessary events to the application (registration success, failure, timeout) as well as the communication process and the data exchange process with another capable entity.

A SIPProxy entity can handle multiple communications at a time.

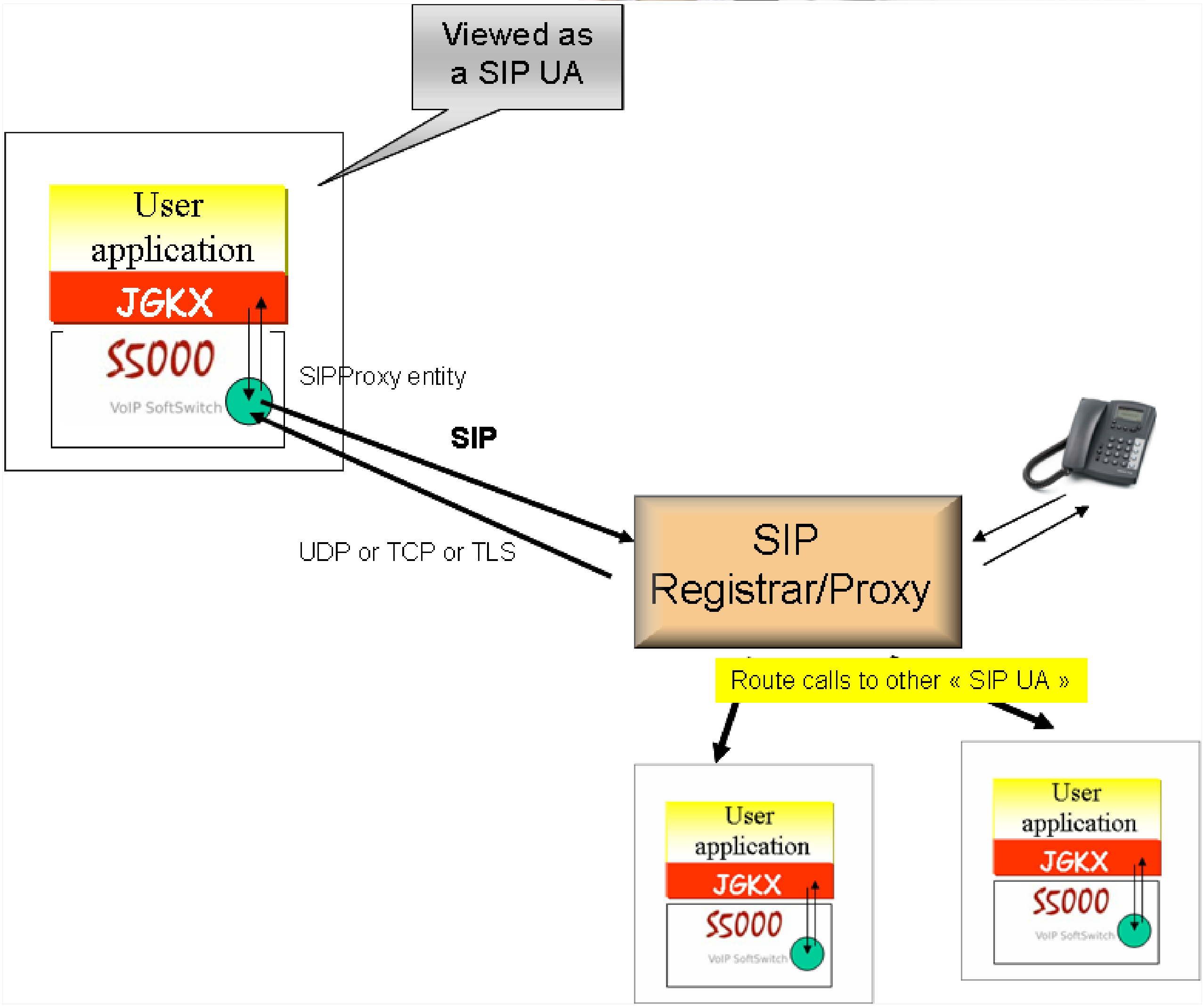


Fig.64 SIP Proxy entity design

Benefit of SIP Stack rich features in a blink

Creating and working with SIP Proxy feature make your application be SIP compliant without the need to understand and handle the detailed network messages involved.
The M2Msoft SIP stack features are used, especially on transport mode: UDP, TCP, TLS can be selected for the SIPProxy.

Free data communication or standardized data communication

Currently, one can transport any data of its choice over the SIP link through the SIP Proxy entity. These data are transported attached to INFO SIP Messages.

Allowed Content types are:

Pure free content	tagged with Content Type set as application/private
CSTA over SIP (ECMA-323)	XML content and is tagged application/csta+xml. Special functions are provided to handle this content

Other non-voice/video contents may be added later on.

This chapter details the functions and events set associated to this SIP Proxy feature.

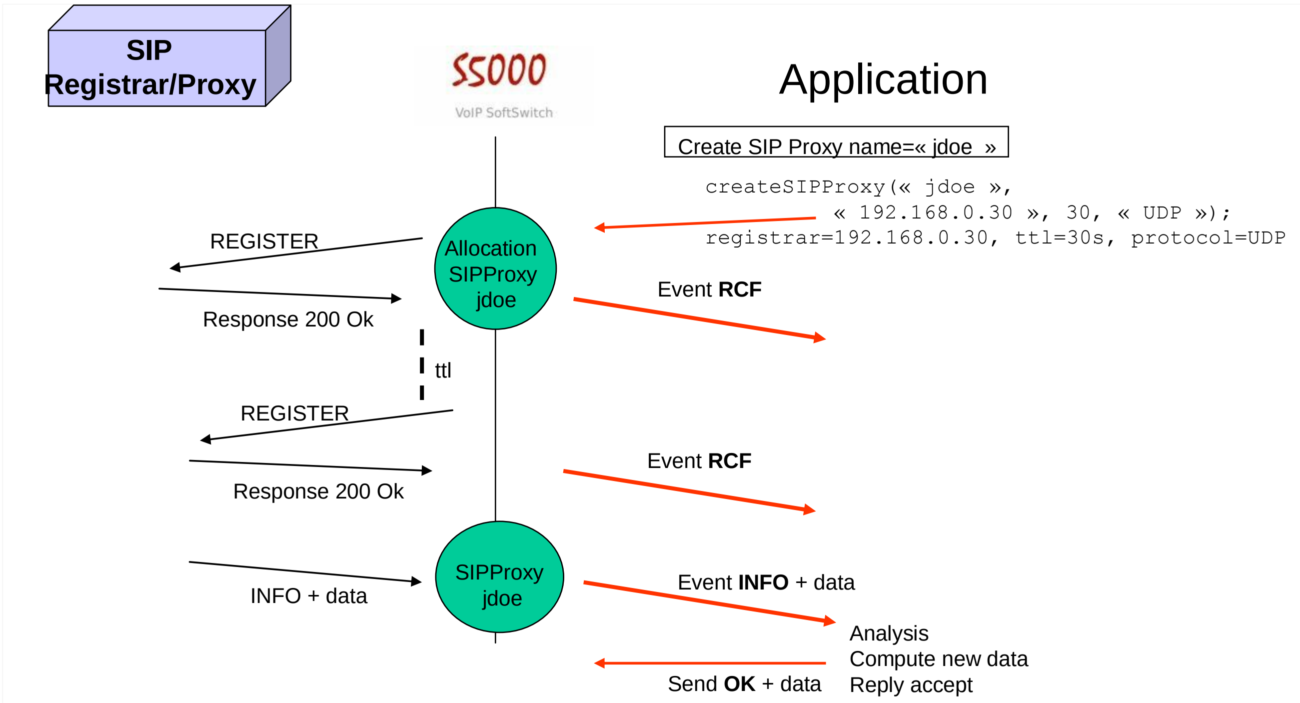


Fig.65 SIP Proxy entity call flow with automatic registration

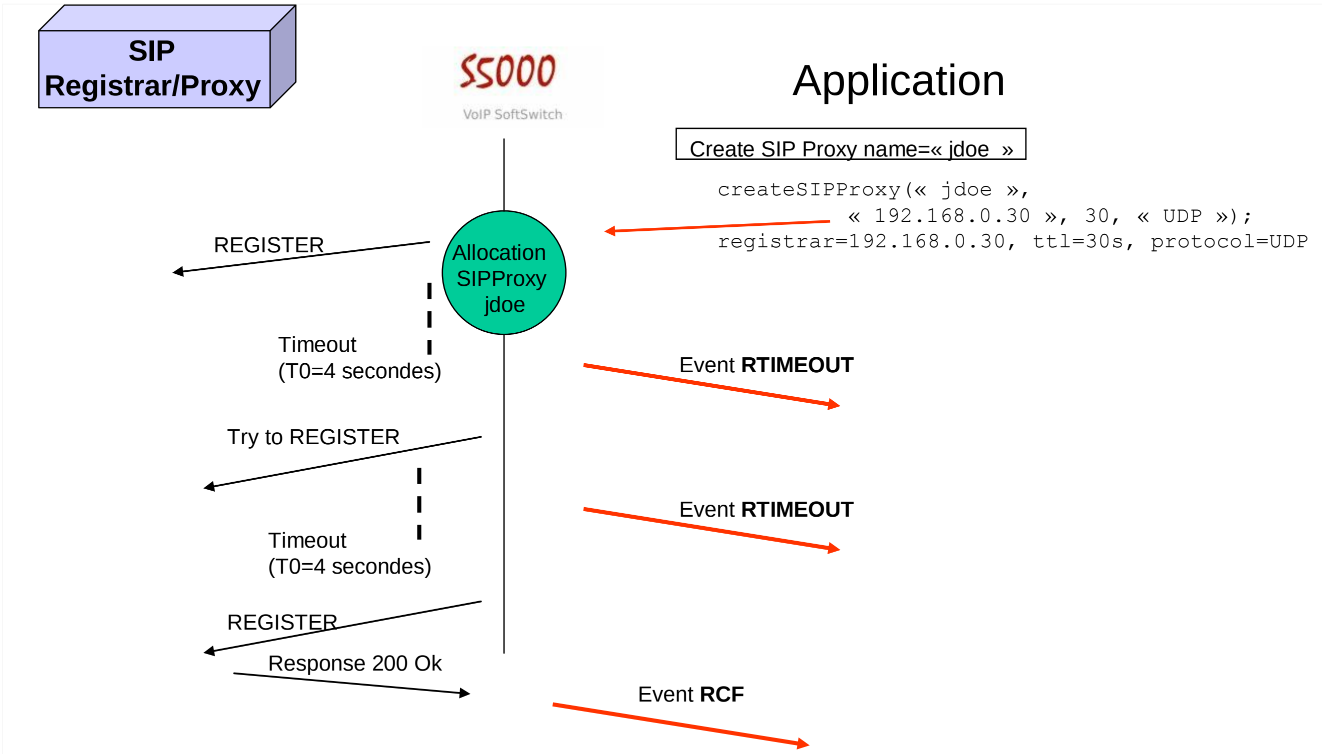


Fig.66 SIP Proxy entity call flow with automatic registration and permanent keep alive

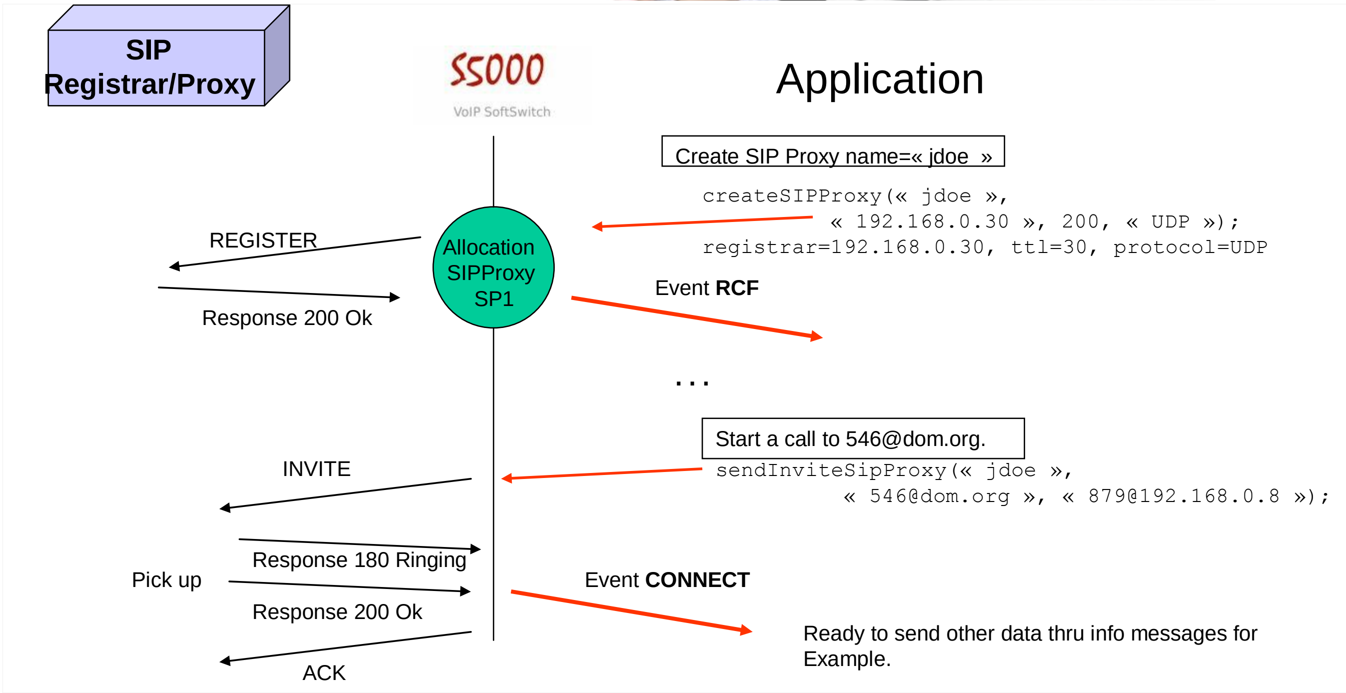


Fig.67 General SIP Proxy entity call flow with an initiated call

Retransmission and timeout timers on unreliable connection links

When operating on UA over unreliable (aka UDP) channels, the SipProxy entity handles the retransmission of requests (INVITE, INFO, BYE). T1 is used as defined per SIP RFC3261. T1 has an internal value of 500 ms.

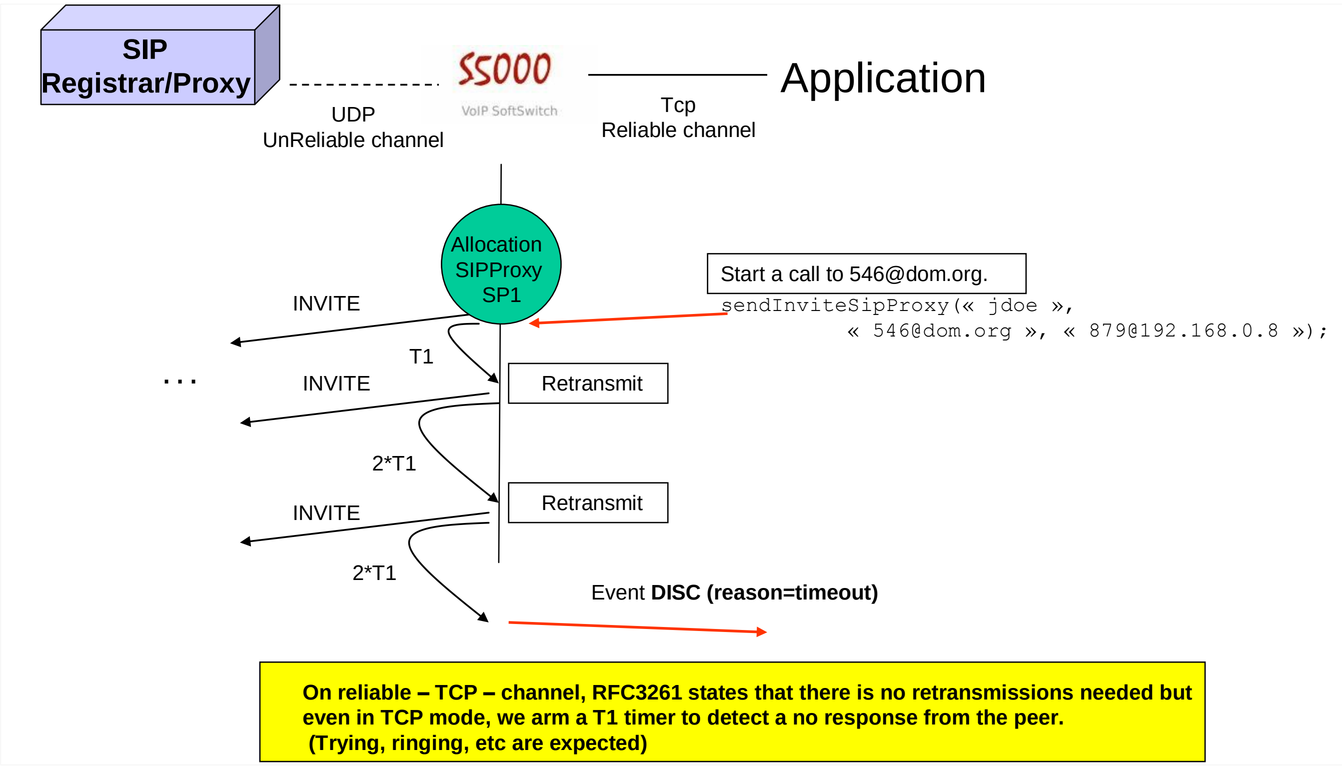


Fig.68 SIP Proxy transport retransmissions handling

The retransmission only applies when no response has been received from peer. If the peer replies with “trying” or “ringing” but does not connect the call, it is up to the application to decide with the appropriate timer, to clear the establishing call, as shown below.

NOTE that sendByeSipProxy generates a CANCEL or a BYE SIP message according to the call status.

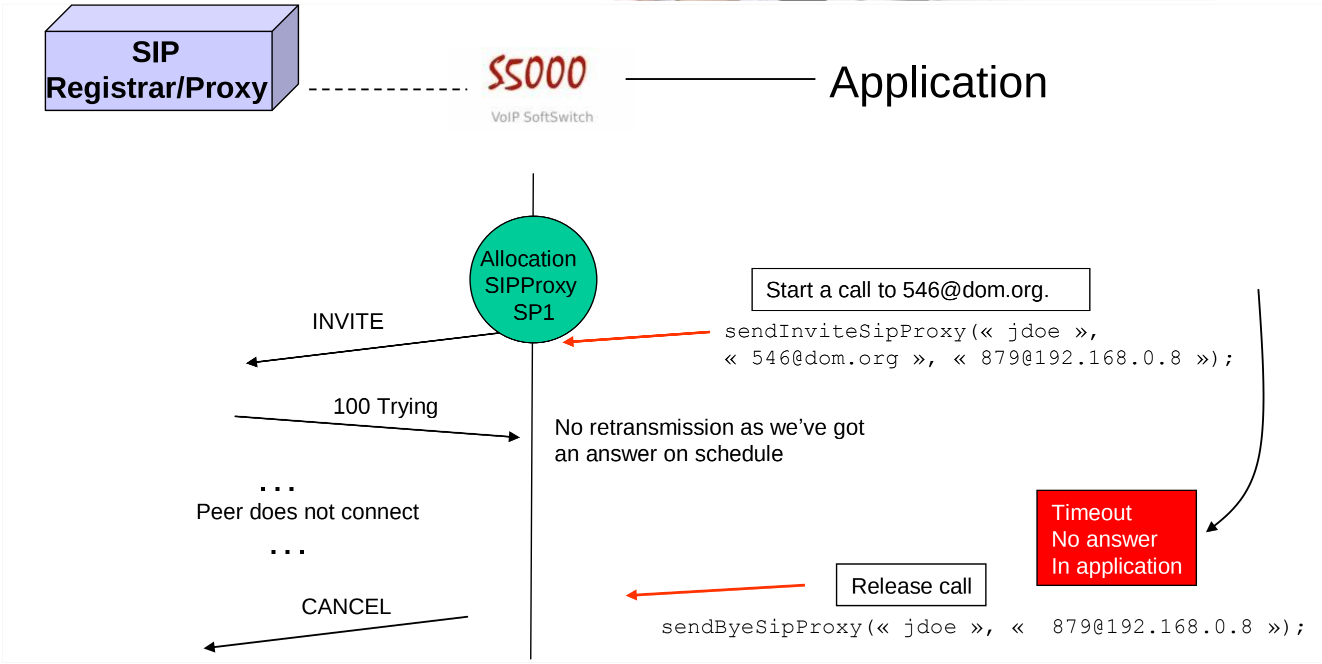


Fig.69 SIP Proxy transport invite with no answer (no connect) from peer.

SipProxy call flows samples

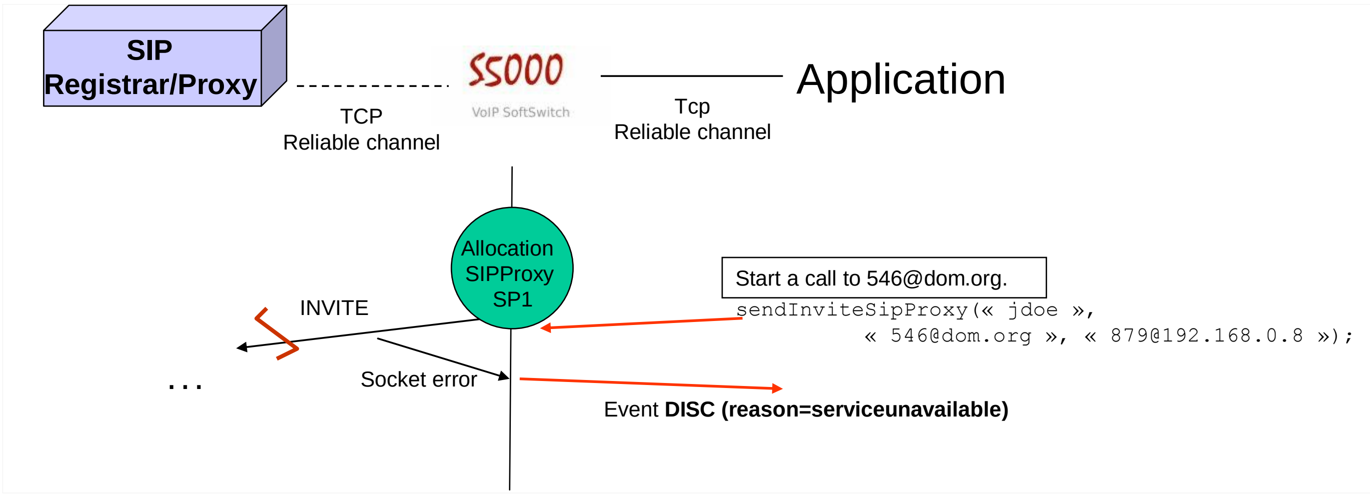


Fig.70 SIP Proxy Invite on socket closed

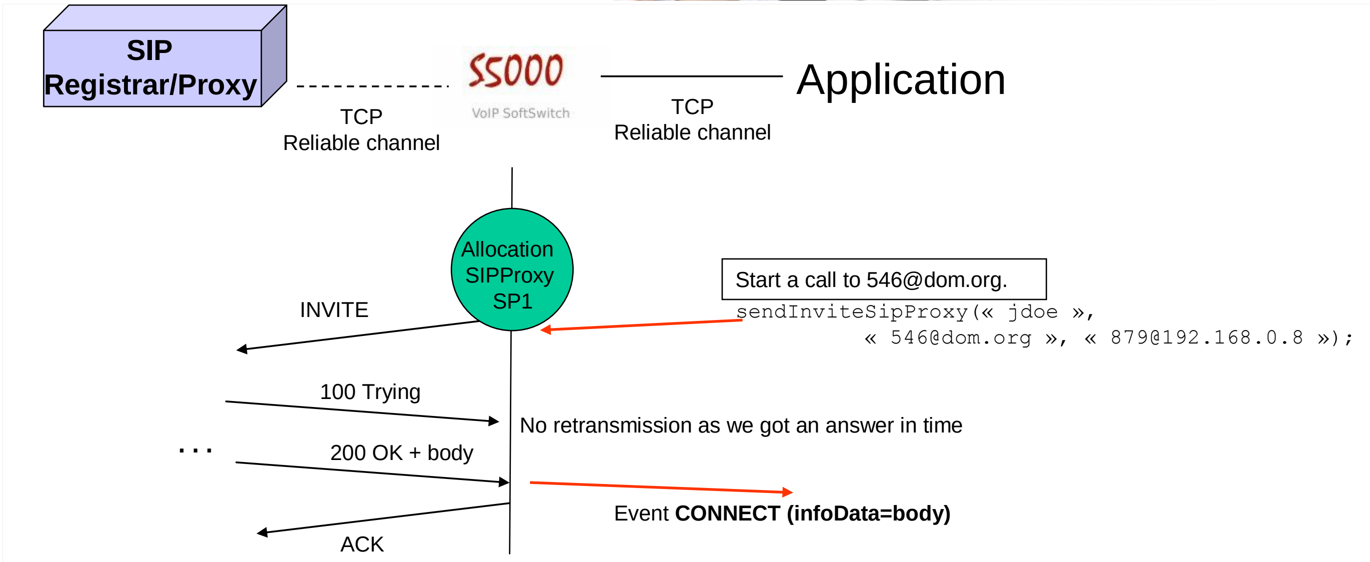


Fig.71 SIP Proxy Invite and call connected

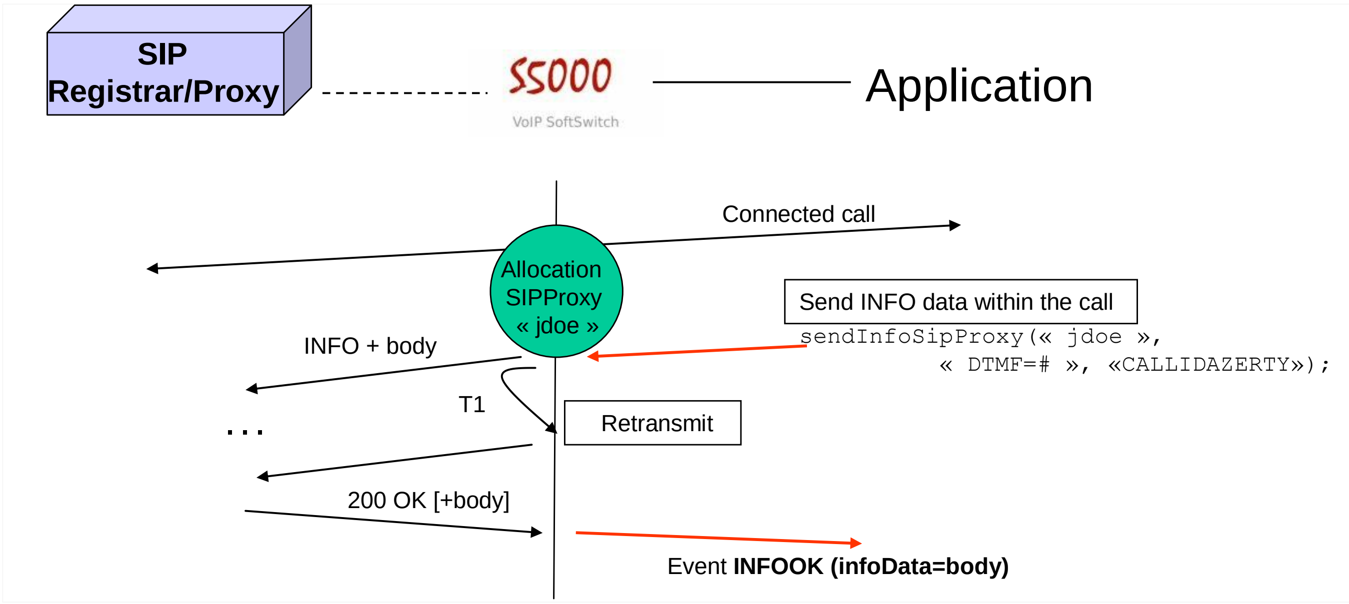


Fig.72 SIP Proxy Info request, with retransmission

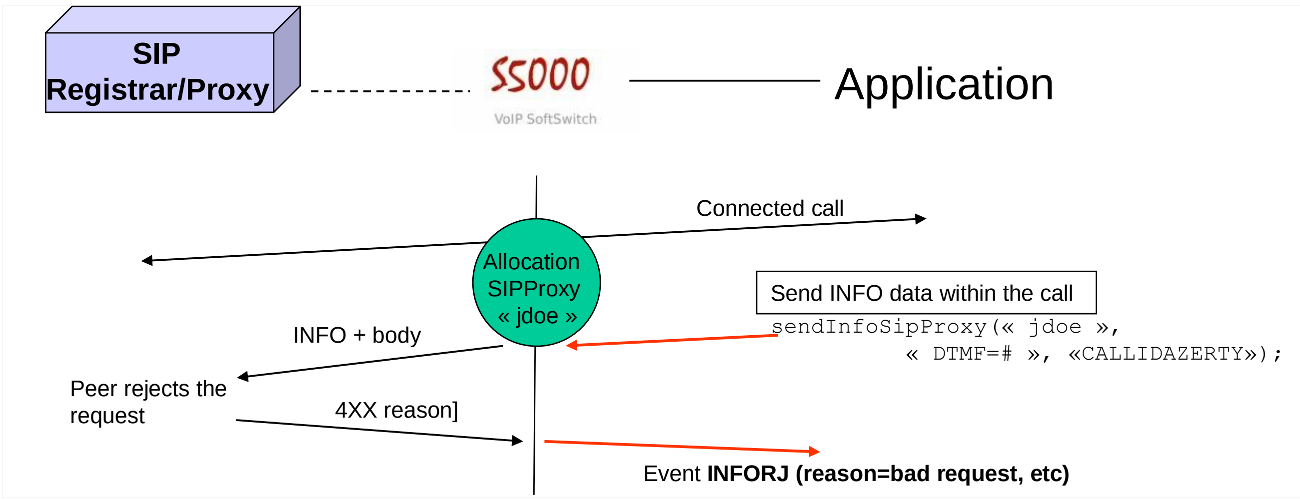


Fig.73 SIP Proxy Info request failure, with error return from party

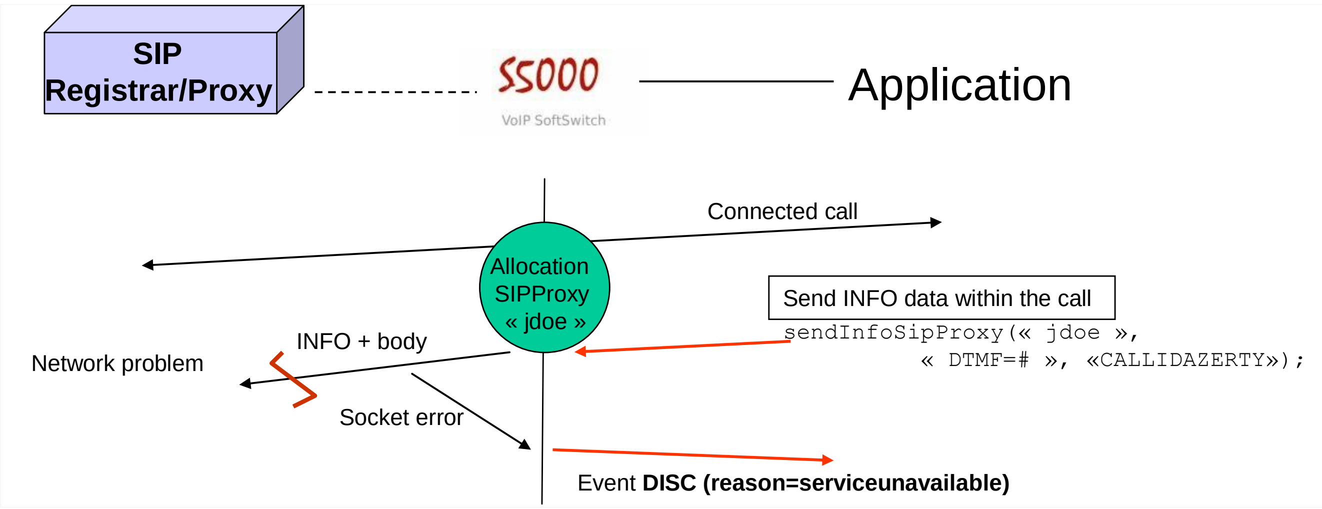


Fig.74 SIP Proxy Info request failure, with network error

7.1.9. What API subset for what usage?

API subset	General API	SIPProxy API	GWControl API	Adm API
What for	Call routing with numbers changes, addresses changes Call termination on standard media (G711/G723.1/G729) Half calls with std media (Gxx codecs) High level management (most protocol messages hidden) H323, SIP independant API	SIP Client development with registration Specific bodies SIP API (CSTA or other text content)	Gateway development with SIP side and standard or specific packets encoding Specific messages bodies Half call with specific body low level management (explicit send of protocol messages) Call contexts rebuild and request for resilience.	Administration Tools API: used to design overall administration, monitoring tools on top of the S5000. For Call Operators, Centrex Operators, etc.

7.2. Java API (JGKXAPI)

7.2.1. How does it work?

The programmer needs the following file to work with:
gimsAPI1.0.jar

This contains all necessary classes to develop and run a client application towards the M2M-S5000 server or any other server based on the JGKXAPI.

The typical structure of a user program is the following:

1The necessary imports from the API
```
/* necessary inclusions*/
import gkcom.*;
import gkcom.jgkx.* ;
```

2The use of a listener Interface
The use of the jgkx Listener enables the processMessage() callback function. This callback function will be called by the API as soon as events occur.
```
public class HelloWorld implements jgkxListener {

// ….
```

3The start function that initialize the API
The jgkx object is created with IP address and TCP port used to connect to the S5000.
The user must:
- create a jgkx instance to connect to the server
- arm the callback function for the events to be caught
- start the job, informing the server to start forwarding events for us

```
String IPADDR=193.7.1.213;
int PORT= 16000;
// connects to M2M-S5000 GK
jgkx cnx = new jgkx(IPADDR, PORT) ;
// add process callback
cnx.addCallback(this);
// set the application name (for routing), request 5 slots and start the main loop
cnx.start(5, "SAMPLE1");
```

4The user process function
In order to receive real time events for all its handled calls, the user must define a call back function named as below in which user code will be placed to act upon the events.
Only one such function is allowed in a client application.
```
public void processMessage(GKMSG msg, CTX ctx)
{
 // user code here with events management

}
```

7.2.2. My HELLO WORD

This chapter details a complete call management application and makes use of the functions and values changes.

Due to the evolving nature of the API, we cannot guarantee that the code depicted here is exactly working with your API version; but we deliver up to date source sample with all our API packages.
Ask your M2MSOFT representative for the working code sample.

Application specifications

MyHelloWorld is an application that performs call deflection upon no answer:

- accept only registrations from endpoints with E164 alias that does not begin with a '2'
- accept calls only to location “999” then forward to a new number : “1003” ; else reject call.

Step by step programming

NOTE: This application works for SIP and/or H323 terminals.

```
package jgkxsample;

/* API includes */
import gkcom.jgkx.*;
import gkcom.*;


/**
 * Client application.
 *
 * Connects to the S5000 server
 *
 */


public class jclient1 implements jgkxListener
{
 public static String IPADDR="193.7.1.216";
 public static int PORT=16000;

 int ctxIdx=0;

 public jclient1()
   {
  entryPoint();
 }

 private void entryPoint()
 {
  System.out.println("entryPoint: entering");
  jgkx api = new jgkx(IPADDR, PORT);
  System.out.println("JGKX API version : "+api.getVersion());

  api.addCallBack(this);
  /* start and request handling of 5 simultaneous calls */
  api.start(5, “Sample1”);
 }
```

```
/* process code HERE !!! */
 public void processMessage(GKMSG gkMsg, CTX ctx)
 {
  System.out.println("processMessage: entering ctx="+ctx);


  /* ******************************************************
   * purpose :
   *  when an endpoint RRQ with 2* => reject
   *  all other are accepted. (RCF)
   * When a called number is 999 => change to 1003 and
      accept (propagate SETUP)
   * else reject (ReleaseComplete generated)
   *************************************************************
   */

  switch (gkMsg.type) {
   case gkcomMsg.RRQ :
    RRQ_REPLY RRQ=new RRQ_REPLY(gkMsg, ctx);

    System.out.println("client1::processMessage: RRQ reply");

    if (gkMsg.e164_source.charAt(0)=='2') {
     System.out.println("client1::processMessage: RRQ reject");
     RRQ.reject();
    }
    else
     RRQ.accept();
   break;

        case gkcomMsg.SETUP :
    SETUP_REPLY SETUP=new SETUP_REPLY(gkMsg, ctx);
    System.out.println("client1::processMessage: SETUP reply");
    if (gkMsg.e164_destination.compareTo("999")==0) {
     SETUP.changeE164Destination("1003");
     SETUP.accept();
     System.out.println("client1::processMessage: SETUP accepted!");
    else
     SETUP.reject();
   break;
  }
  }


  public static void main(String[] args)
   {
  /* start application */
  jclient1 ipapp = new jclient1();
 }
}
```

7.2.3. Packages and Classes

The JGKX API is composed of a file: gimsAPI1.0.jar.
This file contains 2 packages:
gkcom
gkcom.jgkx

These java packages must be imported within every application.

The classes used are:

Class / Interface	Description
jgkx	The main class used to connect to M2M-S5000 and arm callback management.
GKMSG	The event structure passed to the user function. The user can access and modify a number of attributes issued from the H323 or SIP message.
CTX	As a multi-threaded application, a context specific object is handled to the user method as well. **Used to store data and to send commands for the call (Release)**
jgkxListener	Interface to use in every JGKX application. Enable the user class to be handled by the callback system
RRQ_REPLY	Used to activate any change, modify, accept and reject of H323 RRQ or SIP REGISTER event
ARQ_REPLY	Used to activate any change, modify, accept and reject of ARQ event
SETUP_REPLY	Used to activate any change, modify, accept, reject and control of SETUP or SIP INVITE event
CONNECT_REPLY	Used to accept or reject a CONNECT event. Some fields can be set at that time, as the display value.
OLC_REPLY	Used to accept or reject with media channels changes, the H245 channels establishment
OLCACK_REPLY	Used to accept or reject with media channels changes, the H245 channels establishment
SUBSCRIBE_REPLY	Uses to reply to a SUBSCRIBE request. SIP only.

7.2.4. Class jgkx

This class manages the connection with the S5000 and the global internal scheduling of events and callbacks. There are general functions and half calls management functions splitted over four functions subsets.

b) General functions

Method	Signature	Description
jgkx	jgkx(String ip, int tcp_port)	Creates an object to connect to a M2M-S5000 at the ip IP address and tcp_port TCP port. Default value for the port: 16000...
addCallback	void addCallback(jgkxListener obj)	Registers the user main class that contains at least the processMessage() method. This class and method will be used for callback accesses by the API...
start	int start()	Activate the API connection and main loops. As soon as events will arise, the API will automatically call processMessage() within user program. Return -1 when failure to connect to S5K
	int start(int nb)	Activate the API and main loop. Enables the application to handle simultaneously nb calls. *nb* is the total concur. calls being handled and can be 0: in that case, not calls can be handled but only administrative functions may be called. see Administrative functions chapter) Return -1 when failure to connect to S5K
	int start(int nb, String name)	Activate the API main loop and set an application name name. This will be seen with the S5000 for routing. *nb* is the total concurrent calls being handled within this application. *Nb* can be 0. *name* is a unique name to be given to the application (routing can be based on this and inter application communication) Return -1 when failure to connect to S5K
stop	void stop()	Disconnect the current application from S5000, stop the underlying callback.
setForcedRoute	void setForcedRoute(boolean forced)	Call this before start(). When *forced* is true all the call are forwarded to this application, no embeddedService is needed.
setNotifyOnly	void setNotifyOnly(boolean b)	Call this before start(). When setNotifyOnly(true), no call are handled but ALL calls events are monitored in this application. Only administrative functions can be called. (see Administrative functions chapter) Several applications can be in NotifyOnly mode at a time. All NotifyOnly applications receive the events.
getVersion	String getVersion()	The version name and number of the jgkx api used.

addListen	addListen(int pkg)	Call this before start(). Add an event set to be handled within the application. Standard values are : - H245 - RSVP - INFO-CSTA - SUPSERV (H450 events) This adds new events to be handled according to the underlying protocol.
getCallId	String getCallId()	Allocates and returns a unique call identifier (in the H323 meaning). The result String is a human readable string of 32 digits. This value is used as input for the startCall method.

startCall	int startCall(String callId, String calledE164, String destAddr, String callingE164, String display)	Generate a dialout **half communication** / Media Entity. CallId is the callIdentifier of this half call, as generated by getCallId(). CalledE164 is the phone number of the endpoint to call. DestAddr is an IP address to reach as gateway for the call. Use it in case of a voIP Gateway or non-registered recipients. Leave this parameter to “null” in case of a registered recipient. CallingE164 is the mediaEntity to use for this call. Display can be left null or set to a specific value ([A-Za-z0-9] digits) that will be added within the Q931 display element of the call to start. Return 0 when success, -1 if error. In case of a call failure, a DISC event with the related callId will be delivered. In case of success, a SETUP event with the related callId will be delivered. Please refer to [S5KUMAN] for media entities explanations.
joinCall	int joinCall(String mediaEntity1, String mediaEntity2)	Bind two half calls. Enable the RTP/RTCP connection between two **half communications**. MediaEntity1 and mediaEntity2 are the names (aliases) of two valid mediaEntities, connected to endpoints. Connections can be dialin or dialout. Please refer to [S5KUMAN] for media entities explanations.
unjoinCall	int unjoinCall(String mediaEntity1)	Unjoin two **half communications** that were previously joined. Just ask for one of them, mediaEntity1 and the original call is no longer RTP-connected, the parties hear the waiting playfiles associated to their MediaEntities.
setPlayFile	int setPlayFile(String MEName, String fileName)	Set or Change Media Entity playFile. This is immediately activated if MEName is in a call or will be activated for the next call that involves MEName ME.
setMERingDuration	int setMERingDuration(String media, int nb)	Set the ringing delay for the named Media Entity. The ME connects just after nb seconds. Returns -1 if nb <0 or > 65535 Else returns 0.
setMEDisplay	int setMEDisplay(String media, String d)	Set the display d info for the ME media. This display is sent for outgoing calls from this ME or for the connected cqalls to this ME. Rrturns -1 if d is invalid else return 0.

setCodecList	int setCodecList(String MEName, String codecsList)	Set the codecs for the MEName. codecsList is a list of codecs forced to be advertised in H323 H245 TerminalCapabilitySet message and SIP INVITE or 200 OK messages. Priority is defined by the list order, first codec set is first priority. List of codec is expressed as a coma separated string tokens. Up to 9 codecs can be specified. Standard Codecs are defined by the following keywords: G711A G7231 G729 Example of list is; “G711A,G7231” Or “G711A,G729,G7231” Spécial codecs for SIP/SDP protocols are specified by using “SDP<codec definition>” format: Example of list is; “SDPrtpmap:8 PCMA/8000” SDP is the keyword; rtpmap: 8 PCMA/8000 is the parameter. An SDP line according to the parameter set will be produced. Exemple of call: _api.setCodecList("ME1”, "SDPrtpmap:8 PCMA/8000,SDPfmtp 19 ptime=45,G729"); Return >0 if command has been taken. -1 when failure (bad codecs)

setLoop	int setPlayLoop(String MEName, int loopnb)	Defines a number of times to play the Media Entity *MEName* file. *loopnb* can be any number from -1 to 65535. -1 means unlimited. 0 means no file will be played. Return > 0 if command has been taken. -1 when failure (bad loopnb)
create MediaEntity	int createMediaEntity(String MEName)	Creates a MediaEntity named MEName within the S5000. NOTE: setPlayFile() must be used to set a specific audio file to this ME. When success, returns >0. On ME_CREATED event, operation is complete.
splitCall	int splitCall(String callId, String callingME, String calledME)	Split a 'standard' call (point to point) into two halves calls connected to MediaEntities. callId is the callIdentifier of the original established call. callingME is the requested name of the MediaEntity to attach to calling party. calledME is the requested name of the MediaEntity to attach to called party.
getEPList	int getEPList()	Request the current S5000 registered endpoints database view. The event EPLIST is expected. Returns 0 when request has been successfully transmitted, else -1.
getStatus	int getStatus()	Request the current S5000 status upon: License mode, name, description and group status. The event STATUS is expected. Returns 0 when request has been successfully transmitted, else -1.
getCallsList()	int getCallsList()	Request the current S5000 call list (H323, SIP and H323/SIP calls): Call class, callid, caller, called, state, establishing date. The event CALLSLIST is expected. Returns 0 when request has been successfully transmitted, else -1.
sendNotifySIP	int sendNotifySip(String name, String data, String callId, int id)	Talk to a "sip subscriber". Build and generate a spontaneous NOTIFY SIP message to the party "name". (use the h323_source element received within a previous SUBSCRIBE) CallId must match the callId of a previously received SUBSCRIBE event acknowledged. Data will be added as a specific body in the NOTIFY message. Id is unused and can be set to 0. Returns 0 when request has been successfully transmitted, else -1. SIP only.

sendErrorSIP	int sendErrorSip(int code, String data, String callId, int cseq)	SIP protocol specific. Send an error reply message to a “sip user”. Assuming the call is a half call either connected with a mediaEntity or a call “controlled” by the application. (see “generic gateway Controler” chapter) Build and generate a spontaneous <code> SIP error response message to the SIP party. (a sip party connected to the S5000). CallId must match the callId of a previously received SETUP or dialout started. Cseq must be set accordingly to the request one needs to answer withthis error message. Returns 0 when request has been successfully transmitted, else -1. SIP only.
sendIAMessge	int sendIAMessage(String apName, byte[] data)	Inter Application Message. Send a bytes message to an application known by its name *apName*. The recipient application will receive the message within an IAM event. At the recipient application: msg.appName: the name of the originator application msg.appMsgData: the byte data
releaseCall	int releaseCall(String cid, int sipReason, int h323Reason) int releaseCall(String cid)	Releases a call of half call. Cid is the unique call callidentifier. SipReason is an optional SIP release reason (and H323Reason is the H323 release reason) to be taken from chapter 7.2.5 f and g codes list. By default, error is: SIP_BUSY and H323_UNDEFINED For H323, a Q931 releaseComplete is sent to the party or parties. For SIP, depending in the call state, a BYE is sent to the party or parties or a *reason* error is sent.
sendMessage	int sendMessage(String text, String party, String ip)	Sends a short text message to the party phone. The party phone must be registered on the S5000. Party may be in communication or idle. Ip parameter: reserved for future use. The sendMessage() function sends a MESSAGE sip request to the party. (RFC3428) NOTE : for SIP party only.

c) Special SIP Proxy functions

		SIP Proxy methods
createSIPProxy	int createSIPProxy(String name, String alias, String registrar_address, int ttl, int mode, boolean supervised);	Allocates a SIP proxy entity with name *name*, sip uri alias and mode mode. This SIP Proxy will automatically registers to *registrar_address* ip address for *ttl* seconds. If the *ttl*=-1, no registration will be made. The *mode* is: jgkx.UDP or jgkx.TCP. Whne mode is TCP the socket connection can be supervised to detect a tcp failure. (supervised=true). Tcp failure while in a call generates a call disconnection. Return > 0 for command accepted, -1 when immediate failure. The *name* of the SIPProxy will be used in all SIPProxy commands. As the SIPProxy is started, events will come back regularly: RCF, registration accepted, RRJ, registration refused, RTIMEOUT, timeout on registration. Automatic retries are done within the SIPProxy.
haltSIPProxy	int haltSIPProxy(String name)	Frees a SIPProxy previously allocated with name *name*. This frees the internal S5000 resources and unregisters the entity from its registrar. Return > 0 for command accepted or -1 for immediate reject. (bad parameter)

sendInviteSIPProxy	int sendInviteSIPProxy(String name, String remoteAlias, String proxy_address, String cid, String body)	Send a call request. The request is sent to the registrar/proxy where *name* belongs when *proxyIP* is null or to the specified *proxy_address* IP address. The request can attach a text *body*. Nobody is attached when *body* is null. An SDP body will automaticaly be detected as content-type application/sdp. The callrequest is for *remoteAlias* URI, and Call-Id *cid*. (all events will contains this call id) Return > 0 for command accepted or -1 for immediate reject. (bad parameter)
setSIPProxyContentType	void setSIPProxyContentType(String name, String ct)	Set the content type for the data exchanged thru INFO SIP Messages for *name* SIP proxy. Must be set before the call actually connects. Special defines are: jgkx.SIPPROXY_CTYPE_PRIVATE jgkx.SIPPROXY_CTYPE_ECMA323 Other values will be taken inline.
setT1SipProxy	void setT1SipProxy(String name, int t1)	Set a new value for Sip Proxy *name* T1 timer. *T1* is the new value in ms. Default value is : 500 ms Minimum value is : 500 ms
sendInfoSIPProxy	int sendInfoSIPProxy(String name, String body, int cseq)	Send *body* data on an INFO message on the connection. *cseq* is a unique number to correlate INFOOK/INFORKJ events. Return > 0 for command accepted or -1 for immediate reject. (bad parameter)
sendInfoOKSIPProxy	int sendInfoOKSIPProxy(String name, String data, int cseq)	Send data on an OK info message on the connection. *Cseq* is a unique number that must be the one matching the just received INFO. Return > 0 for command accepted or -1 for immediate reject. (bad parameter)
sendInfoERRORSIPProxy	int sendInfoERRORSIPPROXY(String name, int errCode, String data, int cseq)	Send data on an info error message on the current connection (for this named sip proxy). *Cseq* is a unique number that must be the one matching the just received INFO request. Data is an optional body that will be attached to the SIP message. Can be null. Return > 0 for command accepted or -1 for immediate reject. (bad parameter)
sendByeSIPProxy	int sendByeSIPProxy(String name, String cid)	Send a BYE on an established 'callid' connection. Send a CANCEL on an establishing connection. Return > 0 for command accepted or -1 for immediate reject. (bad parameter)

Example of use

```
// create SIP Proxy named 123, that registers and works on SIP over TCP.
// let it send INVITE with SDP data to 5180 user on 192.168.0.101 address.
_api.createSIpProxy("123", host@192.168.0.111, "192.168.0.111", 30, _api.TCP);
String body="V=0\r\n";
Body+="o=61 123456 654343 IN IP4 192.168.0.30\r\ns=none\r\nc=IN
IP4 192.168.0.30\r\nt=0 0\r\nm=audio 10010 RTP/AVP 0
8\r\na=ptime:20\r\na=rtpmap:0 PCMU/8000\r\na=rtpmap:8 PCMA/8000";
_api.sendInviteSIPProxy("123", 5180@192.168.0.101, "192.168.0.111", AZERTY6778",
body);
…
```

Special ECMA 323 CSTA content

For SIPProxy entities working with CSTA/SIP, a set of classes and functions ease the protocol.
In this model, uaCSTA is directly implemented as a B2BUA in the JGKXAPI.
The implementation allows developing uaCSTA aware endpoints or server applications.
See chapter 7.2.17 for details on these classes, to be used in conjunction with SIPProxy entities.

NOTE for compatibility with other endpoints:
When setSIPProxyContentType() is set to SIPPROXY_CTYPE_ECMA323, the INVITE, INFO are sent with:
`Content-Type: application/csta+xml`
`Content-Disposition: signal; handling=required`

In case of a non-support within the remote UA (non uaCSTA endpoint),
a **`415 Unsupported Media Type`** will be expected, thus immediate call release on INVITE. (event DISC within the API)

d) Special Generic Gateway Controler functions

		Generic Gateway Controler methods
startCallExt	int startCallExt (String cid, String called, String data, String calling, String display, String privateData)	Start a generic dial out call. cid is the unique callIdentifier for this call. Called is the called endpoint number. The called number is expected being registered within the S5000. Calling is the calling number. Display (can be null) is a description string that is associated to the calling party and may be displayed on the called screen. Body data is a specific message body (xml for example). Data can be null. SIP usage: the data is used as SIP body for the INVITE. contentType is automatically set according to 'RTP/AVP' found or not. Application/sdp or application/private are set as content-type. H323 : reserved for future use privateData is a short text field to be transmitted to called. No spaces allowed within privateData. privateData can be null. As a command success, a DIALOUT event with the cid callId as set within the startCallExt() is generated. On success, when the dialout has been sent to the party, a DIALOUT event is thrown. On failure, if called party is unknown or (network) unreachable, a DISC event is thrown. Return -1 in case of bad parameter. else return >=0.

	int startCallExt (String cid, String called, String data, String calling, String display,)	Same with null privateData.
sendAlertingExt	int sendAlertingExt(String cid, String data, String privateData)	Send a ringing notification to caller. cid is the callIdentifier, that must exist within the S5000. Data is an optional body. Data may be null. SIP: send a 180 Ringing to calling. H323 : reserved for future use privateData is a short text field to be transmitted to the party. No spaces allowed within privateData. privateData can be null. Return -1 in case of bad parameter. else return code >=0.
	int sendAlertingExt(String cid, String data)	Same with null privateData.
sendConnectExt	int sendConnectExt(String cid, String data, String privateData)	Send a call pickup notification to party. Cid is the unique callidentifier as shown from first message. Dat is a specific message body to end within the call pickup to the caller. Data body may be null. SIP: send a 200 OK INVITE message to caller with specific body. H323 : reserved for future use privateData is a short text field to be transmitted to caller. No spaces allowed within privateData. privateData can be null. Return -1 in case of bad parameter. else return code >=0.
	int sendConnectExt(String cid, String data)	Same with null privateData.

sendReleaseExt	int sendReleaseExt(String cid, String data , String privateData)	Release a call established or establishing. cid is the call callId. Data is a private body to add. Data may be null. SIP: send BYE or CANCEL on the call leg from the S5000. H323: send a Q931 ReleaseComplete on the call leg. privateData is a short text field to be transmitted to party. No spaces allowed within privateData. privateData can be null. DISC event is raised. Return -1 in case of bad parameter. else return code >=0.
	int sendReleaseExt(String cid, String data)	Same with null privateData.
sendErrorExt	int sendErrorExt(int code, String data, String callId, int id, String privateData)	Send an error message on a call leg. SIP: The error response message with sip status ‘code’ is sent. Callid is the call identifier, id will be set for the CSEQ header field and a data body can be attached to the sip message. id can be left to 0, the S5000 manages it automatically. But one can set this precisely if needed. Data may be null in case of no body needed. H323: a call release is done on the call leg with callid call identifier. In this later case, a DISC is raised. id parameter is unused in that case and can be left to 0. Return -1 in case of bad parameter. else return code >=0.
	int sendErrorExt(int code, String data, String callId, int id)	Same with null privateData.
sendNotifyExt	int sendNotifyExt(String name, String data, String callId, int id, String privateData)	Send a message to a subscriber. SIP: send a notify request with callId callidentifier, id CSEQ and a specific data body attached. name is the subscriber number. (use the h323_source element received within a previous SUBSCRIBE) id can be left to 0, the S5000 manages it automatically. But one can set this precisely if needed. Data and name are mandatory parameters and cannot be null. H323: reserved for future use id parameter is unused in that case and can be left to 0.
	int sendNotifyExt(String name, String data, String callId, int id)	Same with null privateData

createCallCtx	int createCallCtx(String callId, String apName, byte state, String calling, String called)	Create a call context within the S5000 and within the named apName application, with the specified callidentifier, state, calling number and called number. State is amongst: GKMSG.CALLCTX_ESTABLISHING GKMSG.CALLCTX_ALERTING GKMSG.CALLCTX_CONNECTED GKMSG.CALLCTX_DISCONNECTED One or both parties must be registered at the time of call. calling number and called number syntax express the call legs protocols : Valid syntax is : sip:<alias>[@control] <alias> is the calling or called number @control : means the call is a half call and the calling or called party is to be handled within the S5000 as if a control() have been made or a startCallExt(). The request is made for an application context that belong to apName application. This application must be running with at least a free context for the command to succeed. Example: *createCallCtx("az8765", "APP2", GKMSG.CALLCTX_ALERTING, "sip:6565", "sip:03456@control");* This defines an INCOMING half call with a SIP calling party with alias(from)=6565 - and it must be registered within the S5000 at that time -, a called number (to)= 03456 and being handled as a half call – due to the @control keyword- within the S5000. *createCallCtx("az8765", "APP2", GKMSG.CALLCTX_CONNECTED, "sip:6565@control", "sip:888");* This defines a DIALOUT half call with a SIP calling party with alias(from)=6565 - and it must be registered within the S5000 at that time -, a called number(to)= 888 and being handled as a half call – due to the @control keyword- within the S5000. The request will be followed asynchronously by a CALLCTX event containing the command result with the S5000. Return 0 is request has been sent successfully, else return -1.

getCallCtx	int getCallCtx(String callId)	Retrieve a call context within the S5000. CallId must match an existing call within the S5000. The request will be followed asynchronously by a CALLCTX event containing the command result with the S5000. Return 0 is request has been sent successfully, else return -1.
delCallCtx	int delCallCtx(String callId)	Delete a call context within the S5000. CallId must match an existing call within the S5000. The request will be followed asynchronously by a CALLCTX event containing the command result with the S5000. Return 0 is request has been sent successfully, else return -1.

Example of use

// Spontaneous start of a dialout call with an sdp body. “0534” calls “1234”.

```
// allocate a callId
 _myCallId= _api.getCallId();
 System.out.println("client1::dialout calId="+_myCallId);
 // defines a sip body

  String body="";
  body+="v=0\r\no=own 001 561 IN IP4 192.168.0.111\r\n";
  body+="s=SIP CALL\r\n";
  body+="c=IN IP4 192.168.0.111\r\n";
  body+="t=0 0\r\nm=audio 50000 RTP/AVP 0 8\r\n";
  body+="a=rtpmap:0 PCMU/8000\r\n";
  body+="a=rtpmap:8 PCMA/8000\r\n";

  _api.startCallExt(_myCallId, /* call identifier */
     "12345",  /* called */
     body,  /* body */
     "0534",   /* calling number*/
     "mydisplay"
    );
```

e) Administration functions

		Administratives methods

getSipAccount	int getSipAccount(
	String name);
		Request informations on a SIP Account element.
		Name is the name of the sip acount as known within the S5000.
		Return SAC event with the list will be raised.
		`Example:`
		`getSipAccount(“LaCie”);`

setSipAccount	int setSipAccount(
	String name, String login, String password, String description,
	String sdaList,
	int pport,
	int maxCalls,
	int maxCallsIN,
	int maxCallsOUT,
	String ufwd, /* uncond fwd*/
	String bud, /*backup dest */
	boolean g729Only,
	boolean noT38,
	String rip, /* restricted IP */
	int kaMax,
	int kaTimer)
		Create a SIP Account with the following fields:
		Name
		Login, *password*, description and list of sda.
		Sdalist is the list of sda number, separated with coma.
		Pport is the preferred port: -1 when no preferred port is requested.
		maxCalls is the maximum allowed concurrent calls (in/out) with the account.
		maxCallsIN is the maximum allowed incoming calls with the account.
		maxCallsOUT is the maximum allowed outgoing concurrent calls with the account.
		Ufwd is the unconditional forward number. If not empty, direct all calls, any time, to this number.
		Bud is the backup sip destination address in case the account is not available at the moment, for incoming calls.
		g729Only: if true, refuse all calls from this account that have other codecs than G729. (forbid all G711 calls for example)
		noT38: if true, refuse all outgoing calls from this account that have T38 codec.
		Rip: only this IP will be accepted for this account callers.
		
		To check a sip account has been created successfully, use the getSipAccount to retrieve the data.
		
		`Example:`
		`setSipAccount("LaCie", "012345", "xc,89", "la cie account", "01303434,01303435", 5089, 50);`

delSipAccount	int delSipAccount(String name);	Request suppression of a sip account within the S5000.
getEPList	void getEPList()	Request the list of all endpoints data within the S5000. Return the EPLIST event with the list will be raised.
getStatus	void getStatus()	Request the S500 state: licensed or not licensed mode. Return the STATUS event with the mode will be raised.
getCallsList	void getCallsList()	Request the list of all calls data within the S5000. Return the CALLSLIST event with the list will be raised.
configSave	int configSave()	Request the S5000 to save its configuration. (gk.ini file is regenerated). Return -1 if request cannot be sent to S5000. Else return 0. CONFSAVED event is returned to indicate the operation result with infoData parameter.

7.2.5. Class GKMSG and events

This class is related to a M2M-S5000 event and stores a set of attributes related to the event.

- **Default event package : (SIP and H323)**
  - RRQ, URQ, ARQ, SETUP, DIALOUT ALERTING, CONNECT, DISC and TIMEOUT.
  - INFO, INFOOK, INFORJ, RCF, RRJ, RTIMEOUT, DISC for SIP Proxies
  - SUBSCRIBE
  - NOTIFYOK, NOTIFYRJ
  - EPLIST, STATUS, CALLSLIST, SAC
  - IAM
  - CONFSAVED
  - CALLCTX

- **H245 event package:**
  - OLC, OLCACK

- **RSVP event package:**
  - RSVP_PATH, RSVP_RESV, RSVP_RESVCONF, RSVP_PATHTEAR

- **MediaEntities event package**
  - ME_MESPLITTED, ME_MECREATED, ME_MECREATERR

- **Supplementary services event package (call transfer, etc.)**
    - SUPSERV (for H450 events)

NOTE: SIP events are mapped on the default event package.

The next table depicts all the S5000 API events from protocol messages.

a) VoIP Signaling events (from Voip events)

SIP message	H323 message	H245 message	RSVP msg	GKX event map
REGISTER	RRQ			**RRQ**
INVITE	SETUP			**SETUP**
BYE, CANCEL	ReleaseComplete			**DISC**
ACK (contextual)	CONNECT			**CONNECT**
Ringing	ALERTING			**ALERTING**
		OpenLogicalChannel		**OLC**
		OpenLogicalChannelAck		**OLCACK**
			Path	**RSVP_PATH**
			Resv	**RSVP_RESV**
			ResvConf	**RSVP_RESVCONF**
			PathTear	**RSVP_PATHTEAR**
			ResvErr	**RSVP_RESVERR**
			PathErr	**RSVP_PATHERR**
INFO				**INFO**
OK on INFO				**INFOOK**
KO on INFO				**INFORJ**
OK on NOTIFY				**NOTIFYOK**
KO on NOTIFY				**NOTIFYRJ**
SUBSCRIBE				**SUBSCRIBE**
OK on REGISTER				**RCF**
KO on REGISTER				**RRJ**
Timeout / register				**RTIMEOUT**
REGISTER with 0 TTL	URQ			**URQ**
unreachable Endpoint	Unreachable Endpoint			**URQ**
	H450			**SUPSERV**
				LOSTCNX (lost S5000 link)
				DIALOUT (a startCallExt() succeed)

Table 1. Signaling events

NOTE: all applications in NotifyOnly mode receive notification of the above VoIP events. There is no need to “accept”, “reject” or “modify” the event as it is notification only. NotificationOnly mode is designed for high precision monitoring applications.

b) Response events (from application commands)

Response events are sent in response to commands.

- Media Entity objects accept commands and the table below shows the awaited events;
- Generic commands expects events in return, as getEPList(). Details follow.

Media Entity command	GKX event	Comment
Creation success Creation error	**ME_MECREATED** **ME_MECREATERR** -privateData field contains the ME name	
Split success Split error	**ME_MESPLITTED** **ME_MESPLITERR**	

<table>
<tr><th>General Command</th><th>GKX event</th><th>Comment</th></tr>
<tr><td>Endpoint list request
getEPList()</td><td>EPLIST
-infoData field contains the endpoint list in an <xml> like view

-aliasList field contains the list of all SIP and H323 endpoint aliases, each-one separated by a coma</td><td>The endpoint list is given as a data string that contains the database description as below, in pseudo BNF:

EPList: entrylist END
entrylist: entry SEP entrylist | entry
entry: <class=classval ; type=typeval ; alias=aliasval ;contactAliases=contactAliasesVal; ipp=ippval; ttl=ttlval;info=infoval >

NOTE: parameters can come in any order.

classval: H323 | SIP
typeval: integer (reserved)
aliasval: string | string , aliasval
contactAliasesVal: string, string, …
ttlval: integer
infoval: string
ippval: ip_address COLON portval
portval : integer
ip_address : integer . integer . integer. integer (IPV4)

COLON: :
SEP: \n
END: \n\n</td></tr>
<tr><td>S5000 status request
getStatus()</td><td>STATUS
-infoData field contains the status list in an <xml> like view</td><td>The status is given as a data string that contains the database description as below, in pseudo BNF:

STATUS: < stvarlist >
stvarlist: entry ; stvarlist | entry
entry: <version=vers ; name=nameval ; licstatus=liststatus ; groupstatus=grpval;master=yes/no >

vers: 1.83-rxx
nameval: S5000 name
licstatus: "no license" | version-intermediate
grpval: true|false
masterval: true|false

SEP: \n
END: \n\n</td></tr>
</table>

S5000 Calls List request getCallsList()	**CALLSLIST** -infoData field contains the status list in an <xml> like view	The call list is given as a data string that contains the database description as below, in pseudo BNF: **CALLSLIST:** < callvarlist > END **callvarlist:** entry SEP callvarlist | entry **entry:** <class=*classval* ; cid=string ; state=*stateval* ; caller=string; [sacaller=*accountname*;] called=string; [sacalled=*accountname*;] establishingDate=date > *classval*: H323Call | SIPCall |H323SIPCall *stateval*: ESTABLISHING | ALERTING | CONNECTED *accountname*: string, optional element, sip account name (if not present, the party does not have any associated account) caller: calling party alias called: called party alias SEP: \n END: \n\n
S5000 Sip Account data request getSipAccount()	**SAC** -infoData field contains the sip account data in an <xml> like view	The call list is given as a data string that contains the database description as below, in pseudo BNF: <class=SipAccount ; name=string ; result > END **result:** login=string ; password=string; description=string; maxCalls=int; maxCallsIn=int; maxCallsOut=int | "non existent" ;noT38=boolean boolean: true / false END: \n\n
S5000 configuration save request. confSave()	**CONFSAVED** -infoData field contains the operation result.	infoData = "success" or "failure" (config file could not be written for example)
S5000 call context operation. createCallCtx(), getCallCtx(), delCallCtx()	**CALLCTX** -infoData field contains the operation result.	InfoData contains the context command result. <operation=opval;callid=callId;result=resval[;reason=reasonval]> **opval:** create | delete |get **resval:** success | failure **reasonval:** cannot find a slot | unsupported | sip party not registered

End points list EPLIST infoData field format

Entry Attribute	**class** (String)	**type** (integer)	**aliases** (String)	**ttl** (integer)	**info** (String)	**Ipp** (String)
Description	H323 or SIP	For future use	Endpoint known aliases, E164, H323-ID, URI; coma separated values	Time to live as known (may change)	Product and vendor information as extracted from the endpoint messages	Ip address and port of the terminal. (ras address for H323, SIP address from requests in SIP)

```
Example:
<class=H323;type=1;ipp=0.1.10.12:1726;aliases=60005,SIEMNS;ttl=15;info="Hin
et LP5100">
<class=SIP;type=0;ipp=192.168.0.30:5060;alias=5100@192.168.0.30;5060,Office
;contactAliasesVal=5101,5102;ttl=2;info="Swissvoice IP10S">
\n
\n
```

Calls list CALLSLIST infoData field format

Entry Attribute	**class** (String)	**cid** (String)	**caller** (String)	**sacaller (string)**	**called** (String)	**sacalled (string)**
Description	H323Call or SIPCall or SIPH323Call	Call id of the call (as set by the caller)	originator known number (sip: name@domain)	Optional Sip account for caller	Called party known number (sip: name@domain)	Optional Sip account for called

Entry Attribute	**Esta. Date** (string)	**state** (String)
Description	Date of call setup Day_of_week month day year hour:min:sec:ms	Call state ESTABLISHING ALERTING CONNECTED

Example:

```
<class=SIPCall;cid=536c-c0a80101-0-
2@192.168.0.44;caller=5144@192.168.0.30;sacaller=CIE1;called=5145@192.168.0
.30;sacalled=MYCIE;establishingDate=Sun May 06 2007
17:25:09.257;state=CONNECTED>
<class=SIPH323Call;cid=4142434445464748494A4B3130303034;sipParty=5146@192.1
68.0.30;h323Party=911;establishingDate=Sun May 06 2007
17:25:57.738;state=CONNECTED>
\n
```

Sip account data SAC infoData field format

Entry Attribute	**class** (String)	**name** (String)	**login** (String)	**Password** (String)	**description** (string)	**maxCalls** (int)	**maxCallsIn** (int)	**maxCallsOut** (int)
Description	SipAccount	Name of the account	login	password	Account desc.	Total conc. calls alowed	Total IN conc. calls alowed	Total OUT conc. calls allowed

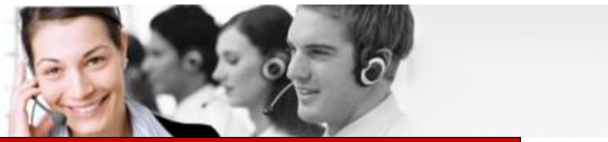
Example:
```
<class=SipAccount;name=prTel;login=prt;password=partner;description=special
_account;maxCalls=30;maxCallsIn=30;maxCallsOut=30>\n\n
```

c) GKMSG methods

Method	Signature	Description
getInformation TransferCapability()	int getInformationTransferCapability ()	Returns an abstract from the Bearer capability of a call that indicates the audio or audio/video nature of the call. Valid values are: SPEECH AUDIO31KHZ UDIGITAL (unrestricted digital) RDIGITAL (restricted digital) The UDIGITAL and RDIGITAL values are most often associated with audio+Video calls. H323 only.

d) GKMSG attributes

Attributes	Type	Description
transaction	String	A unique number allocated within the gatekeeper. Read Only
type	int	The message type (event) amongst the values : gkcomMsg.RRQ gkcomMsg.URQ gkcomMsg.ARQ gkcomMsg.SETUP gkcomMsg.ALERTING gkcomMsg.CONNECT gkcomMsg.DISC gkcomMsg.TIMEOUT gkcomMsg.SUPSERV gkcomMsg.SUBSCRIBE gkcomMsg.INFO gkcomMsg.INFOOK gkcomMsg.INFORJ gkcomMsg.RTIMEOUT gkcomMsg.DIALOUT gkcomMsg.EPLIST gkcomMsg.STATUS gkcomMsg.CALLSLIST gkcomMsg.SAC Read Only
sessionId	String	The slot number for this application, where the event occurred Read Only
ip_source	String	The IP address of the caller



Attributes	Type	Description
ip_dest	String	The IP address of the recipient endoint
e164_source	String	The E164 alias from the originator endpoint (if any E164 is provided) Read/Write – see later on for change
e164_dest	String	The E164 alias for the called recipient.
h323_source	String	The H323Id alias from the originator endpoint (if any H323Id is provided)(also the from field of SIP messages)
h323_dest	String	The H323Id Alias – if any- for the called endpoint
privateData	String	A private data area that can be forwarded thru special H323/Q931 or SIP fields. Unlimited length. Transport is done thru: -NonStandardParameter field of Q931 SETUP message; -Generic parameter of SIP INVITE Contact field Subject to change in subsequent versions, contact your local dealer for this Only with SETUP messages. Read/Write – see later on for change
ttl	int	The Time To Live in seconds as advertised by endpoint in RRQ/REGISTER event.
bandwidth	String	Only in ARQ messages for H323. The bandwidth value as requested by the caller terminal. The value is expressed in 100th. For example: 1280 means a 128 KB is requested. Read/Write
callId	String	The callIdentifier unique value per call as allocated by the H323 sender terminal. This value will be set with ARQ, SETUP, ALETING, CONNECT, RELEASE, OLC, OLCACK messages. Read Only
display	String	The Q931 DISPLAY information element. Present in SETUP, CONNECT messages. Read/Write – see later for change
isStatic	boolean	True when the message came from a static entity element within the S5000. False otherwise: call comes from a registered endpoint.
gateway	boolean	True when the message came from a Gateway system. False otherwise. Only with SETUP messages. Read only.
ip_ifDestAddress	String	In Multihomed platform, contains the local ip address that received the event. Present in all events. Read Only
h245MediaChannel	String	RTP address for the media channel Only with OLC and OLCACK Read/Write

Attributes	Type	Description
h245MediaPort	int	RTP UDP port for the media channel Only with OLC and OLCACK Read/Write
h245MediaControl	String	RTCP address for the control channel Only with OLC and OLCACK Read/Write
h245MediaControl Port	int	RTCP UDPport Only with OLC and OLCACK Read/Write
h245Session	int	Channel number (to separate audio, video) Only with OLC and OLCACK Read/Write
h245DataType	int	Codec type AUDIODATA VIDEODATA APPLICATIONDATA Only with OLC and OLCACK
h323Reason	int	Protocol detailed reason for call release. H323 Reason. (see tables 3 and 4 below)
sipReason	int	Protocol detailed reason for call release. SIP Reason (see tables 3 and 4 below)
rsvpSession	String	RSVP session information (unique string info with IP address and port, to all RSVP exchanges for one rsvp controlled flow)
rsvpTimeValues	String	This RSVP message refresh timeout (ms)
rsvpSenderTemplate	String	Srce address and Src port (UDP here)
rsvpStyle	String	Shared Explicit (“SE”), Fixed Filter (“FF”) or Wildcard Filter (“WF”)
rsvpFlowSpec	String	Int serv information: token bucket
rsvpFilterSpec	String	Same as sender template
infoData	Sring	SIP messages : Private or application specific data body contained within INFO, INFOOK or REGISTER message S5000 messages : EPLIST event data.
cseq	String	SIP messages only. The CSEQ SIP header attribute value. Example: “3 INVITE” This cseq value can be used to build user defined messages to endpoints.
locallyInitiated	boolean	For DISC events only. True when the DISC results from a sendReleaseExt() of the application.

Attributes	Type	Description
appName	String	IAM message. On IAM Inter Application Messaging, name of the originator application.
appMsgData	byte[]	IAM message. On IAM Inter Application Messaging, data content of the message received.
aliasList	String	Filled after an EPLIST or RRQ event. Contains the list of all SIP and H323 endpoint aliases. Each one is separated from the others by a coma. Only the aliases (without domain) are printed.

Table 2 – Events Attributes list

NOTE: If an element was not in the original event, the value is empty.
It is recommended to the developer to test against null values.

e) Attributes per event

The tables below depict all the managed events and the expected attributes to be read.

M: Mandatory
O: Optional/ May not be present
S: SIP only (when event is generic)
H: H323 only (when event is generic)
*: at least one information is present amongst all O (*)

Attribute	RRQ	URQ	ARQ	SETUP	ALERTING	CONNECT	DISC	OLC	OLCACK
transaction	M	M	M	M	M	M	M	M	M
type	M	M	M	M	M	M	M	M	M
sessionId	M	M	M	M	M	M	M	M	M
ip_source	O	M		M				M	
ip_dest				O					
e164_source	O (*)	O (*)	O	O (*)				O (*)	
e164_dest			O	M (*)					
h323_source	O (*)	O (*)	O	O (*)				O (*)	
h323_dest			O	M (*)					
privateData				O					
bandwidth			M						
callId			O	M	M	M	M	M	M
display				O		O			
gateway			M	MH					
ip_ifDestAddress				M				M	
h245Address				OH		OH			
h245MediaChannel								O (*)	O (*)
h245MediaPort								O(*)	O (*)
h245MediaControl								O (*)	O (*)
H245MediaControlPort								O (*)	O (*)
H245Session								M	M
H245DataType								M	M
h323Reason							OH		
sipReason							OS		
infoData	O			OS		OS			
cseq	MS	MS		MS	MS	MS	MS		
locallyInitiated							MS		
aliasList	M								
data	M	M			M	M			

Attribute	PATH	RESV	RESVCONF	PATHTEAR	RESVERR	PATHERR
rsvpSession	M	M	M	M	M	M
rsvpTimeValues	M	M				
rsvpSenderTemplate	M					M
rsvpStyle		M	M		M	
rsvpFlowSpec		O(*)	O(*)		O*	
rsvpFilterSpec		O(*)	O(*)		O*	

Attribute	EPLIST	SUPSERV	SUBSCRIBE
transaction	M	M	M
Type	M	M	M
sessionId	M	M	M
callId		M	M
infoData	M		O
aliasList	M		
supservOperation		M (h450 operation: gkMsg. CALLTRANSFERIDENTIFY CALLTRANSFERINITIATE CALLREROUTING_H4503 DIVERTINGLEGINFO1_H4503 DIVERTINGLEGINFO2_H4503 DIVERTINGLEGINFO3_H4503 DIVERTINGLEGINFO4_H4503 HOLDNOTIFIC_H4504 RETRIEVENOTIFIC_H4504 REMOTEHOLD_H4504 REMOTERETRIEVE_H4504)	
e164_dest		M (rerouting number)	
h323_source			O
h323_dest			M
data		O	O

Attribute	INFO	INFOOK	INFORJ	RTIMEOUT	DIALOUT	IAM	CALLCTX
infoData	O	O			O		M
callId	M	M	M	M	M		
h323_source	M	M	M	M	M		
e164_destination					M		
sipReason			M	M			
cseq	M	M	M		M		
appName						M	
appDataMsg						M	
Data	O						

Attribute	ME_CREATED	ME_CREATERR
transaction	M	M
Type	M	M
sessionId	M	M
privateData	M MediaEntity name)	M (Media Entity name)

f) H323 Release reasons

In case of call reject, the table below lists predefined H323 Release reason codes that can be set or compared.
The reason is set and get from H225.0 message element from Q931 messages emitted/received.
This is contained within msg.h323Reason variable.

H323 Reason codes
H323_NOBANDWIDTH
H323_RESOURCESEXHAUSTED
H323_UNREACHABLEDESTINATION
H323_DESTINATIONREJECTION
H323_INVALIDREVISION
H323_NOPERMISSION
H323_UNREACHABLEGK
H323_GATEWAYRESOURCE
H323_BADFORMATADDRESS
H323_UNDEFINEDREASON
H323_FACILITYCALLDEFLECTION
H323_SECURITYDENIED
H323_CALLEDPARTYNOTREGISTERED
H323_CALLERNOTREGISTERED
H323_NEEDEDFEATURENOTSUPPORTED

Table 3 – H323 reason codes

g) SIP Release reasons

In case of call reject, the table below lists predefined SIP Release reason codes that can be set or compared. This is contained within msg.sipReason variable.

SIP Reason codes for Error responses to INVITE
SIP_BADREQUEST
SIP_UNAUTHORIZED
SIP_PAYMENTREQUIRED
SIP_FORBIDDEN
SIP_NOTFOUND
SIP_METHODNOTALLOWED
SIP_NOTACCEPTABLE
SIP_PROXYAUTHREQUIRED
SIP_REQUESTTIMEOUT
SIP_UNSUPPORTEDMEDIATYPE
SIP_TEMPORARYUNAVAILABLE
SIP_ADDRESSINCOMPLETE
SIP_BUSYHERE
SIP_REQUESTPENDING
SIP_SERVERINTERNALERROR
SIP_NOTIMPLEMENTED
SIP_SERVICEUNAVAILABLE
SIP_SERVERTIMEOUT
SIP_VERSIONNOTSUPPORTED

Table 4 – SIP reason codes

h) Examples of use

RSVP

```
/*… RSVP event analysis */
public void processMessage (GKMSG msg, CTX ctx)
{
/*…*/

case gkcomMsg.RSVP_PATH:
   System.out.println("RSVP path with session="+msg.rsvpSession);
break;
/*…*/
```

H323

```
case gkcomMsg.SETUP:
   int itc=msg.getInformationTransferCapability();
   if (itc != GKMSG.SPEECH && itc!=GKMSG.AUDIO3KHZ)
      System.out.println("SETUP received for an audio/video call !");
   else
      System.out.println("SETUP received for an audio only call !");

break;
```

H323 Or SIP

```
// rejects 5151 named terminal. Accepts all others.
case gkcomMsg.RRQ:  /* got a RRQ or a REGISTER */
   if (msg.e164_source.compareTo("5151")==0)
      RRQ.reject();
   else
      RRQ.accept();
break;
```

7.2.6. Class CTX

This class is related to a M2M-S5000 session of events and is passed to the user service function as a convenient way to access his private information relative to a session/slot.
As the JGKX applications are multi-threaded ones, the user is able to store and then retrieve any information he would like associated to every call context.
The CTX allows the user to activate/deactivate a timer dedicated to the slot it has been activated.

In the current JGKX version, a single timer per slot can be activated. As much as the number of slots concurrent timers can be activated in a single user application.
In the current JGKX version, the number of slots per application is defined at start (start() method) or set to a default value : 2.

In a call situation (i.e. for the ARQ/SETUP/CONNECT/DISC period), CTX is passed accordingly to the GK slot used.
The CTX object can be used to act on any call handled within the application:

- forward a call to a new location
- release a call
- transfer a call to a new location

Method	Signature	Description
setData	void setData(Object o)	The user stores a set of data, o, within this context for later retrieval.
getData	Object getData()	The user is able to retrieve any information set he might have store previously with a setData().
setTimer	void setTimer(int duration)	Activates a timer for the duration seconds. At the expiration of the timer, a TIMEOUT event is generated according to the slot (CTX object) where the timer was created. ProcessMessage() is called with a TIMEOUT event and the CTX object. The timer is automatically removed after the event.
RemoveTimer	void removeTimer()	Removes a timer previously created with setTimer() and before it has expired.
releaseCall	void releaseCall(String callId)	Release order for an ongoing call with callId callIdentifier. This makes the M2M-S5000-GK to send a ReleaseComplete message to the call initiator. The callId value must be taken out of ARQ, SETUP, ALERT or CONNECT messages.

forwardCall	void forwardCall(String newCalled)	Release a current NON CONNECTED call, and re-establish a new connection with a new called endpoint. This can be used in conjunction with a timer to handle Forward on no answer actions. NOTE: This function can be used to handle ACD like call distribution on a group of numbers.
transferCall	void transferCall(String callingParty, String calledParty)	Transfer the CTX associated call to a new called party. The initial call must be connected. CallingParty is the CTX call party to keep in the new call context. CalledParty is the called endpoint alias. CTX reflects the new call between callingparty and calledParty. The user must manage the new call events. (not available in all versions)

Attributes	Type	Description
none		

Example of use

```
if (msg.ip_source.compareTo("10.0.0.8")==0 {
   // store data
   USER_CLASS obj = new USER_CLASS();
   obj.my_data="I have seen the address of  GW1";
   ctx.setData(obj);
}

// …. Later on
if (msg.type==gkcomMsg.SETUP) {
   // retrieve any  data
   obj = (USER_CLASS)(ctx.getData());
   System.out.println("data="+obj.my_data);
}
```

7.2.7. Class SETUP_REPLY

This class is used to build a SETUP answer that M2M-S5000 will use for its actions.
A SETUP_REPLY object can only be built while in a SETUP event processing.
The SETUP_REPLY object contains all the information for the resulting (and hence modified) SETUP message M2M-S5000 might forward to the other party.
A SETUP_REPLY object contains all the initial SETUP message values and the developer can modify some of these via object methods. For example, one can modify the display element with .changeDisplay() method.

NOTE: a SETUP event is thrown for H323-SETUP or SIP-INVITE messages.

Method	Signature	Description
changeDisplay	void changeDisplay(String s)	Modify the the Q931 Display information element for the SETUP to be forwarded. If the display value was empty, this is used to set the value s.
changeDestination	void changeE164Destination(String s)	Modify the called alias. **The new called number can be a media entity** (to create a half call).
changeSource	void changeE164Source(String s)	Modify the callingNumber aliases. Aliases may b entered separated with a ‘,’ Example: “12345,John” Will create a E164 alias=12345 and an H323Id alias =”john”.
changeIPDestination	void changeIPDestination(String s)	Modify the destination IP address to send the call. This can be used to direct a Gateway or another Gatekeeper or any H323 terminal IP address may be entered with : <ip address>:<tcp_port> Example: 193.7.1.210:1721
setPrivateData	void setPrivateData(String s)	Used to set user data that does not interfere with the underlying protocol but are conveyed to the final recipient. A proper application can then take out and use these data.

setNoAutoconnect	void setNoAutoconnect()	This is used **only in case of media Entity** routing, to disable the automatic connect of the media entity after the alerting. With this option, the caller is no longer connected and hear the ringing tone until the application decides to connect this half call.
setAudioOnly	void setAudioOnly()	Force a call without video and t120 data capabilities. Only audio capabilities are kept in H245 mode...
setRSVP	void setRSVP(boolean m, int ttl, int callleg)	Activate/disable RSVP mode for the current call. Ttl is the refresh period for the rsvp path. The callleg is how to apply rsvp on one or both parties of the call. An rsvp managed call takes place between parties and the S5000 in the middle. This defines two call legs. Call leg is set with: CALLING_LEG (set on the source) CALLED_LEG (set on the destination) CALLEDANDCALLING_LEG (rsvp both sides)
setH323ReleaseReason	void setH323ReleaseReason(int code)	Set an H323 release Reason code (see table 3) to be returned to the H323 calling party. (for SIP party, use the setSIPReleaseReason()) *NOTE: Both setSip… and setH323… can be set at any time, no matter what endpoint type if calling.*
setSipReleaseReason	void setSipReleaseReason(int code)	Set a SIP error code (see table 4) to be returned to the SIP calling party. (for H323 party, use the setH323ReleaseReason()). *NOTE: Both setSip… and setH323… can be set at any time, no matter what endpoint type if calling.*
accept	void accept()	Send the response to S5000 and accept the request
reject	void reject()	Send the response to S5000 and reject the request. H323 endpoint: a ReleaseComplete message with no special reason is sent to caller. SIP endpoint: A “Forbidden” ERROR Message is returned to caller.

control	void control()	Let the application control the call. The S5000 will not process the call. (as this is the case with accept() or reject()) The application must later on use sendAlertingExt(), sendConnectExt(), etc to proceed this call side. Note: For security reason, the S5000 internally kills calls that arer establishing for too long. If no action (alerting, connect) is made on an incoming call within the application after the control() for too long, the call will be released.
	void control(int delay)	Same as previous but with a grace delay on the call allowing it to be establishing for the supplementary *delay* in seconds. control(300); Let the call being stopped for 5 minutes in case no action is done.
redirect	void redirect (int code, String newNum)	Send the response to the S5000 for this call: ask for call redirect to another num. A SIP answer with code “code” is directed to the caller asking for a new call on alias “newNum”. Example: redirect(302, “56”); Recommended Code values: GKMSG.SIP_MULTIPLECHOICES GKMSG.SIP_MOVEDPERMANENTLY GKMSG.SIP_MOVEDTEMPORARILY GKMSG.SIP_USEPROXY GKMSG.SIP_ALTERNATESERVICE

Attributes	Type	Description
None		

Example of use

```
/*…*/
case gkcomMsg.SETUP:
   SETUP_REPLY sr=new SETUP_REPLY(msg, ctx);
   if (msg.e164_destination.compareTo(“1215”)==0 {
      // reject this call
      sr.reject();
   }
   else {
      sr.changeE164Source(“0123456”); // change calling number
      sr.accept(); // let the call proceed
   }
}
```

Class RRQ_REPLY

This class is used to build a RRQ answer that M2M-S5000 will use for its actions.
A RRQ_REPLY object can only be built while in a RRQ event processing.
The application can choose to reject the endpoint registration request at that point with a reject() or to continue the registration with an accept().
When the decision is to continue the registration, one can set/modify the endpoint elements with changeE164() method for example to force a set of dynamic aliases.

NOTE: An RRQ event is thrown for H323-RRQ or SIP-REGISTER messages.

Method	Signature	Description
changeE164	void changeE164(String s)	Replace the first E164 alias found with this one. The endpoint will act as if this dynamic alias had been set in first place.
changeH323	void changeH323Id(String s)	Replace the first H323Id alias found with this one. The endpoint will act as if this dynamic alias had been set in first place.
addAlias	void addAlias(String s)	Add a new alias to this endpoint. Accoring to the alias syntax, it will be autiloatically set in the form of E164 or H323Id.
setInfoData	void setInfoData(String body)	SIP only Set a specific body to REGISTER response. (can be 200 or another)
setSipErrorCode	void setSipErrorCode(int c)	SIP only Set a specific response message code. C is an integer 0-16384. setSipErrorCode(200); means a 200 OK will be replied to the party. Any other value can be set.
accept	void accept()	Send the response to S5000 and accept the registration.
reject	void reject()	Reject the registration.

Attributes	Type	Description
none		

7.2.8. Class ARQ_REPLY

This class is used to build an ARQ answer that M2M-S5000 will use for its actions.
An ARQ_REPLY object can only be built while in a ARQ event processing.

NOTE: only used while receiving H323-ARQ message.

Method	Signature	Description
changeBandwidth	void changeBandwidth(int bw)	Modify the the call bandwidth. In /100 of bit. Example, changeBandwidth(1280) means 128 000 bits bandwidth requested.

Attributes	Type	Description
none		

7.2.9. Class CONNECT_REPLY

Pertain to the default event package.
This class is used to build a CONNECT answer that M2M-S5000 will use for its actions.
A CONNECT_REPLY object can only be built while in a CONNECT event processing.
The application can choose to stop the call at that point with a **reject**() - a ReleaseComplete H323 will be generated in the H323 call legs- or to continue the call with an **accept**() or avoid the S5000 doing any action by calling **control**() - Generic Gateway Controler API -.
When the decision is to continue the call, one can set/modify the display element with .changeDisplay() method in order to display useful information to the caller.
When the decision is to control the call, the Generic Gateway Controler functions must be called later on for ringing, connect, release the call, etc.

Method	Signature	Description
changeDisplay	void changeDisplay(String s)	Modify the the Q931 Display information element for the CONNECT to be forwarded. If the display value was empty, this is used to set the value s.
setJoinCallId	Void setJoinCallId(String callId)	In case of **half call management**, use this to request a connect to be forwarded to a waiting half call (see SETUP_REPLY). As soon as the two half calls have opened their media channels, they are joined and talked together. CallId is the H323-CallIdentifier of the other call to connect to. This can have been saved just before a startCall() or within the SETUP event of this other call.
accept	void accept()	Send the response to S5000 and proceed the connect. H323 and SIP.
reject	void reject()	Make the S5000 rejecting the call. The caller and called parties are released. H323 and SIP.
control	void control()	Let the application control the call. The S5000 will not process the call. (as this is the case with accept() or reject()) The application must later on use sendAlertingExt(), sendConnectExt(), etc to proceed this call side. H323 and SIP.

Attributes	Type	Description
none		

7.2.10. Class OLC_REPLY

This is to be used within the H245 events package. The developer must add a jgkx.addEventPackage(H245) to activate this event.
This class is used to build an OpenLogicalChannel answer that M2M-S5000 will use for its actions.
An OLC_REPLY object can only be built while in an OLC event processing.
The application can modify the media/RTP/RTCP addresses at that point.

Method	Signature	Description
changeChannelAddress	void changeChannelAddress (String addr)	Modify the the RTP and RTPC addresses
changeMediaChannelPort	void changeMediaChannelPort (int p)	Modify the the RTP port. The p port must be even. The p+1 port will be set for the RTCP channel.

Attributes	Type	Description
none		

7.2.11. Class OLCACK_REPLY

This is to be used within the H245 events package. The developer must add a jgkx.addEventPackage(H245) to activate this event.
This class is used to build an OpenLOgicalChannelAck answer that M2M-S5000 will use for its actions.
An OLCACK_REPLY object can only be built while in a OLCACK event processing.
The application can modify the advertised media/RTP/RTCP addresses at that point. The media route within the network can be completely controlled and separated from the signaling flow with these commands.

Method	Signature	Description
changeChannelAddress	void changeChannelAddress (String addr)	Modify the the RTP and RTPC addresses
changeMediaChannelPort	void changeMediaChannelPort (int p)	Modify the the RTP port. The p port must be even. The p+1 port will be set for the RTCP channel.

Attributes	Type	Description
none		

7.2.12. Class SUBSCRIBE_REPLY

Specific SIP signaling event response.
Pertain to the default event package.
This is used to acknowledge a SUBSCRIBE event.
A SUBSCRIBE_REPLY object can only be built while in a SUBSCRIBE event processing.

Method	Signature	Description
accept	void accept()	Send the response to S5000: the subscription is registered within the S5000 and an OK message sent to the originator.
reject	void reject()	Send the response to S5000 and instruct not to ignore the request.

Attributes	Type	Description
none		

7.2.13. Class templateEmbedService

This class is used to build or modify embedded service. All fields of embedded service are represented as private attributes of templateService

Method	Signature	Description
getName	String getName()	Accessors allow to get /set "**Name**" of embedded service
setName	void setName(String name)	
getSourceMask	String getSourceMask()	Accessors allow to get /set "**Source Mask** Mask" of embedded service
setSourceMask	void setSourceMask(String mask)	
getSourceFwd	String getSourceFwd()	Accessors allow to get /set "**Forwarded source**" of embedded service
setSourceFwd	void setSourceFwd(String fwdSrce)	
getDestinationMask	String getDestinationMask()	Accessors allow to get /set "**Destination Mask**" of embedded service
setDestinantionMask	void setdestinationMask(String fwd)	
getDestinationFwd	String getDestinationFwd()	Accessors allow to get /set "**Forwarded destination**" of embedded service
setDestinationFwd	void setDestinationFwd(String fwd)	
getAppName	String getAppName()	Accessors allow to get /set "**Application name**" of embedded service
setAppName	void setAppName(String name_appli)	
getSipAccount	String getSipAccount()	Accessors allow to get /set "**Sip account mask**" of embedded service
setSipAccount	void setSipAccoun(String accMask)	
getTYPE	String getTYPE()	Accessors allow to get /set "**Type**" of embedded service
setTYPE	void setTYPE(String type)	
getCodecList	String getCodecList()	Accessors allow to get /set "**Sip Codecs mask**" of embedded service
setCodecList	void setCodecList(String codecs)	
getRoute	String getRoute()	Accessors allow to get /set "**Route** " of embedded service
setRoute	void setRoute(String route_name)	
getDestADDR	String getDestADDR()	Accessors allow to get /set "**Target**" of embedded service
setDestADDR	void setDestADDR(String addr)	
getMediaFile	String getMediaFile()	Accessors allow to get /set "**Media file**" of embedded service
setMediaFile	void setmediaFile(String media_nom)	
getPosition	int getPosition()	Accessors allow to get /set "**Position**" of embedded service
setPosition	void setPosition(int pos)	

7.2.14. Class templateRoute

This class is used to build or modify route. All fields of route are represented as private attributes of templateRoute

Method	Signature	Description
getName	String getName()	Accessors allow to get /set "**Route name**" of route
setName	void setName(String name)	
getListTrunks	String getListTrunks()	Accessors allow to get /set "list of **Trunks list** " of route
setListTrunks	void setListTrunks(String trink_list)	
getMode	String getMode()	Accessors allow to get /set "**Trunks mode**" of route
setMode	void setMode(String mode)	
getCodeErr	String getCodeErr()	Accessors allow to get /set "**Err.codes to cont.**" of route
setCodeErr	void setCodeErr(String err)	

7.2.15. Class templateTrunk

This class is used to build or modify trunk. All fileds of trunk are represented as private attributes of templateTrunk

Method	Signature	Description
getName	String getName()	Accessors allow to get /set "**Trunk name** " of trunk
setName	void setName(String name)	
getTarget	String getTarget()	Accessors allow to get /set "**Targets**" of trunk
setTarget	void setTarget(String targets)	
getCodecMask	String getCodecMask()	Accessors allow to get /set "**Codecs filtering**" of trunk
setCodecMask	void setcedecMask(String codec)	
getTimer	int getTimer()	Accessors allow to get /set "**No-Answer timer [sec] (0=infinite)**" of trunk
setTimer	void setTimer(int time)	
getAlgo	String getAlgo()	Accessors allow to get /set "**Algorithm**" of trunk
setAlgo	void setAlgo(String algo)	
getMaxCalls	int getMaxCalls()	Accessors allow to get /set "**Max call (Empty for unlimited)**" of trunk
setMaxCalls	void setMaxCalls(int mxCalls)	
getNewDestination	String getNewDestination()	Accessors allow to get /set "**New destination alias**" of trunk
setNewDestination	void setNewcalls(String newdest)	

7.2.16. Class templateProvision

This class is used to build or modify endpoint Profile. All fields of endpoint are represented as private attributes of templateProvision

Method	Signature	Description
getName	String getName()	Accessors allow to get /set "**Name**" of endpoint
setName	void setName(String name)	
getAlias	String getAlias()	Accessors allow to get /set "**Alias**" of endpoint
setAlias	void setAlias(String alias)	
getMac	String getMac()	Accessors allow to get /set "**Mac address**" of endpoint
setMac	void setMac(String mac)	
getDisplay	String getDisplay()	Accessors allow to get /set "**Display**" of endpoint
setDisplay	void setDisplay(String display)	
getPhoneType	String getTypePhone()	Accessors allow to get /set "**Auto provision**" of endpoint
setPhoneType	void setTypePhone(String type)	
getRestrictions	String getRestrictions()	Accessors allow to get /set "**Dialout Restrictions**" of endpoint
setRestrictions	void setRestrictions(String rest)	
getIPAddr	String getIPAddr()	Accessors allow to get /set "**IP addr (if static)**" of endpoint
setIPAddr	void setIPAddr(String ip)	
getMask	String getMask()	Accessors allow to get /set "**Mask (if static)**" of endpoint
setMask	void setMask(String mask)	
getGateway	String getGateway()	Accessors allow to get /set "**Gateway (if static)** " of endpoint
setGateway	void seGateway(String gateway)	
getForwardType	String getForwardType()	Accessors allow to get /set "**Forward type**" of endpoint
setForwardType	void setForwardType()	
getForwardNumber	String getForwardNumber()	Accessors allow to get /set "**Forward to other destination**" field of endpoint
setForwardNumber	void setForwardNumber(String num)	
getForwardMsg	boolean getForwardMsg()	Accessors allow to get /set "**Forward to messaging**" field of endpoint
setForwardMsg	void setForwardMsg()	
getForwardTime	String getForwardTime()	Accessors allow to get /set "**Forward NoAnswer timer**" field of endpoint
setForwardTime	void setForwardTime(int time)	

Method	Signature	Description
getCTone	String getCTone()	Accessors allow to get /set "**Country Tone**" of endpoint
setCTone	void setCTone(String tone)	
getLanguage	int getLanguage()	Accessors allow to get /set "**Language**" of endpoint
setLanguage	void setLanguage(int lang)	
getEthCx	int getEthCx()	Accessors allow to get /set "**Ethernet connection**" of endpoint
setEthCx	void setEthCx(int eth)	
getAccountLogin	String getAccountLogin()	Accessors allow to get /set "**Sip authentication login** " of endpoint
setAccountLogin	void setAccountLogin(String login)	
getAccountPass	String getAccountPass()	Accessors allow to get /set "**Sip authentication password**" of endpoint
setAccountPass	void setAccountPass(String pass)	
getLines	int getLines()	Accessors allow to get /set "**Number of lines**" of endpoint
setLines	void setLines(int line)	
getSipCallingPrefix	String getSipCallingPrefix()	Accessors allow to get /set "**SipCallingPrefix for xfe**" field of endpoint
setSipCallingPrefix	void setSipcallingPrefix(String prefix)	
getSharedLines	String getSharedLines()	Accessors allow to get /set "**Shared line**" of endpoint
setSharedLines	void setSharedLines(String line)	
getFunctionTouches	String getFunctionTouches()	Accessors allow to get /set "**F1,F2…**" of endpoint
setFunctionTouches	void setFunctionTouches(String touch)	
getWebPass	String getWebPass()	Accessors allow to get /set "**Web user password**" of endpoint
setWebPass	void setWebPass(String pass)	
getPtime	int getPtime()	Accessors allow to get /set "**Forced G729** " of endpoint
setPtime	void setPtime(int ptime)	
getDistingushTone	boolen getDistingushTone()	Accessors allow to get /set "**Distinguished melody for external calls**" of endpoint
setDistingushTone	void setDistingushTone(boolean tone)	
getSupervisedCall	boolean getSupervisedCall()	Accessors allow to get /set "**Sip call supervised**" of endpoint
setSupervisedCall	void setSupervisedCall(boolean super)	
getTimeZone	int getTimeZone()	Accessors allow to get /set "**Timezone**" of endpoint
setTimeZone	void setTimeZone(int time)	

Method	Signature	Description
getDhcp	boolean getDhcp()	Accessors allow to get /set “**DHCP**” of endpoint
setDhcp	void setDhcp(boolean d)	
getVlan	boolean getVlan()	Accessors allow to get /set “**VLANs** ” of endpoint
setVlan	void setVlan(boolean vl)	
getVlanVoice	int getVlanVoice()	Accessors allow to get /set “**Voice**” of endpoint
setVlanVoice	void setVlanVoice(int voice)	
getVlanData	int getVlanData()	Accessors allow to get /set “**Data**” of endpoint
setVlanData	void setVlanData(int data)	
getCallWaitTone	boolean getCallWaitton,e()	Accessors allow to get /set “**Call Waiting Tone disabled**” of endpoint
setCallWaitTone	void setCallWaittone(boolean w)	
getGroup	String getGroup()	Accessors allow to get /set “**Group**” of endpoint
setGroup	void setGroup(String gr)	
getMelody	int getMelody()	Accessors allow to get /set “**Melody**” of endpoint
setMelody	void setMelody(int melo)	

7.2.17. ECMA-323 Package

For SIPProxy entities working with CSTA/SIP, a set of classes and functions ease the protocol.
In this model, uaCSTA is directly implemented as a B2BUA in the JGKXAPI.
The implementation allows to develop uaCSTA aware endpoints or server applications.

The JGKXAPI comprises a special java package to manage the uaCSTA XML elements in requests and responses over SIP, com.m2msoft.uaCSTA.

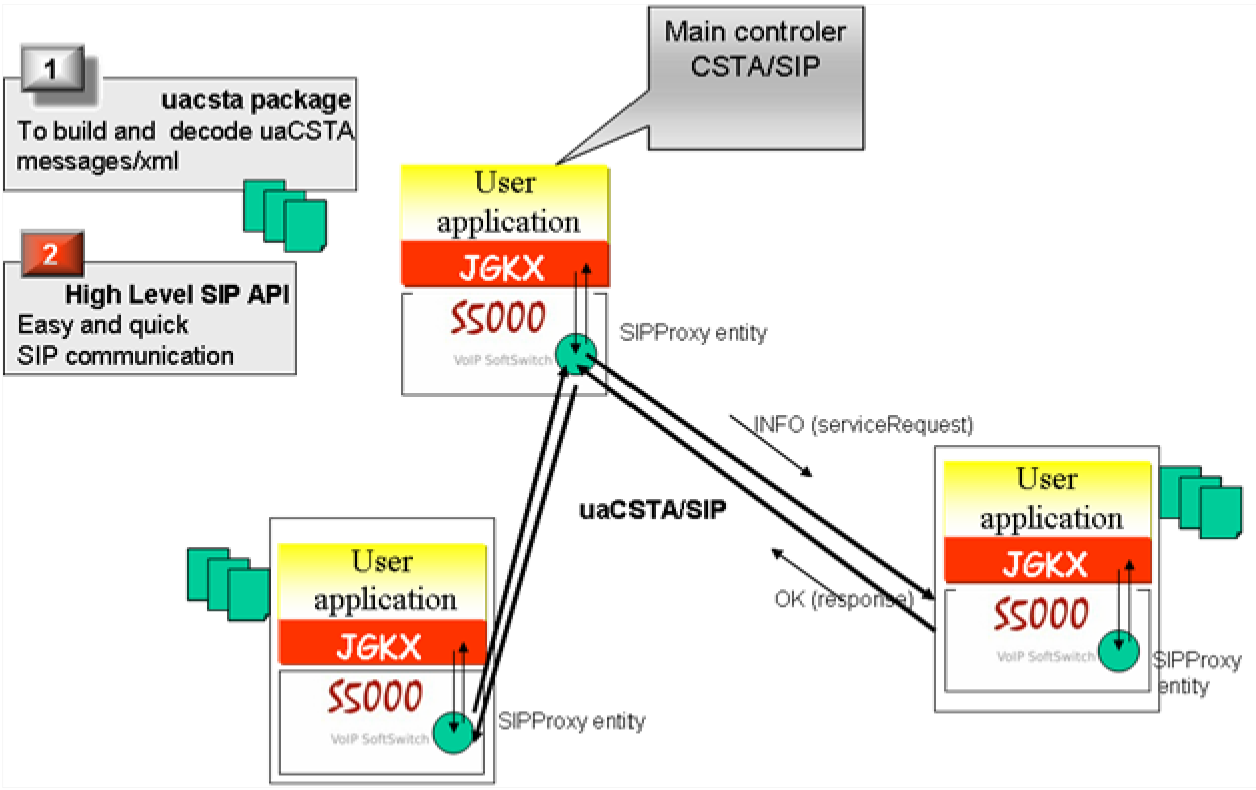


Fig.75 ua CSTA package use for PABX communication

The following classes apply on the transported data on the INFO methods.

The table below shows the uaCSTA message groups and the support level in the M2Msoft uaCSTA package.

Group	Example	Supported
Call Control	ClearConnection, …	Yes
Physical Phone Features	GetSpeakerVolume, …	No
Logical Phone Features	GetDoNotDisturb, …	No
Monitoring Services & Events	MonitorStart, …	Yes – Only voice can be set
Snapshot services	SnapshotDevice, …	No
Discovery & System status	Get CSTA Features	No

Classes and attributes are named as the ECMA-323 standard with ‘Request’ and ‘Response’ appended. A generic CSTAError class is defined for the negative responses.
The classes contain simple attributes for the elementary unique elements and object attributes for the structured fields. These lead to additional classes within the package.

Package com.m2msoft.uaCSTA

CSTAMessage (Interface)
All subsequent messages implements this interface.

Method	Signature	Description
build	String build()	Encode an AlternateCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Call
A generic class to store call info. This is used in call control requests and responses

Method	Signature	Description
build	String build()	Encode an AlternateCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_callId	String	M	See [ECMA]
_deviceID	String	M	See [ECMA]

CSTAErrorResponse
A generic class for negative responses.

Method	Signature	Description
build	String build()	Encode an AlternateCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_operation	String	M	See [ECMA]

Package com.m2msoft.uaCSTA.callcontrol

<u>AlternateCallRequest</u>
This is used to build or decode an AlternateCallRequest message to hold/retrieve an existing call.

Method	Signature	Description
build	String build()	Encode an AlternateCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_heldCall	Call	M	See [ECMA]
_activeCall	Call	M	See [ECMA]

<u>AlternateCallResponse</u>
This is used to build or decode an AlternateCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an AlternateCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
none			

<u>AnswerCallRequest</u>
This is used to build or decode an AnswerCallRequest message to answer an alerting call.

Method	Signature	Description
build	String build()	Encode an AlternateCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object . Return false when decoding fails

Attributes	Type	M/O	Description
_callToBeAnswered	Call	M	See [ECMA]

AnswerCallResponse
This is used to build or decode an AnswerCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an AlternateCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
none			

ClearConnectionRequest
This is used to build or decode a ClearConnectionRequest message to clear acall.

Method	Signature	Description
build	String build()	Encode an AlternateCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_connectionToBeCleared	Call	M	See [ECMA]

ClearConnectionResponse
This is used to build or decode an AnswerCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an AlternateCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
none			

ConsultationCallRequest
This is used to build or decode a ConsultationCallRequest message to hold a current call and start a new one.

Method	Signature	Description
build	String build()	Encode a ConsultationCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_existingCall	Call	M	Call to put on hold
_consultedDevice	String	M	New number to dial

ConsultationCallResponse
This is used to build or decode a ConsultationCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an ConsultationCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_initiatedCall	Call	M	The new call parameters

DeflectCallRequest
This is used to build or decode a DeflectCallRequest message to move a call top a different destination.

Method	Signature	Description
build	String build()	Encode a DeflectCall structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_callToBeDiverted	Call	M	Call to be diverted
_newDestination	String	M	New sip uri

DeflectCallResponse
This is used to build or decode an DeflectCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an ConsultationCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none	Call	M	The new call parameters

GenerateDigitsRequest
This is used to build or decode a GenerateDigitsRequest message to send DTMF on a call from the UA.

Method	Signature	Description
build	String build()	Encode a GenerateDigitRequest structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_connectionToSendDigits	Call	M	Call to apply action
_charactersToSend	String	M	DTMF string to send

GenerateDigitsResponse
This is used to build or decode an GenerateDigitsResponse, positive response.

Method	Signature	Description
build	String build()	Encode an ConsultationCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

HoldCallRequest
This is used to build or decode a HoldCallRequest message to put a call on hold at a UA.

Method	Signature	Description
build	String build()	Encode a HoldCallRequest structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_callToBeHeld	Call	M	Call to apply action

HoldCallResponse
This is used to build or decode a HoldCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an HoldCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

MakeCallRequest
This is used to build or decode a MakeCallRequest message to start a call from this UA.

Method	Signature	Description
build	String build()	Encode a MakeCallRequest structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_callingDevice	String	M	SIP URI of calling
_calledDevice	String	M	SIP URI of called device
autoOriginate	int	M	PROMPT, DONOTPROMPT

MakeCallResponse
This is used to build or decode an GenerateDigitsResponse, positive response.

Method	Signature	Description
build	String build()	Encode an MakeCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_callingDevice	Call	M	

ReconnectCallRequest
This is used to build or decode a ReconnectCallRequest message to clear a specified call and retrieves a held call at the UA.

Method	Signature	Description
build	String build()	Encode a ReconnectCallRequest structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_activeCall	Call	M	Call to clear
_heldCall	Call	M	Call to retrieve

ReconnectCallResponse
This is used to build or decode a ReconnectCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode an ConsultationCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

RetrieveCallRequest
This is used to build or decode a RetrieveCallRequest message to retrieve a call that was on hold.

Method	Signature	Description
build	String build()	Encode a RetrieveCallRequest structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_callToBeRetrieved	Call	M	Call to apply action

RetrieveCallResponse
This is used to build or decode a RetrieveCallResponse positive response.

Method	Signature	Description
build	String build()	Encode a RetrieveCallResponse structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

SingleStepTransferCallRequest
This is used to build or decode a SingleStepTransferCallRequest message to transfer a call from the UA to a party.

Method	Signature	Description
build	String build()	Encode the request structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_activeCall	Call	M	Call to transfer
_transferedTo	String	M	New number, sip uri or E164

SingleStepTransferCallResponse
This is used to build or decode an SingleStepTransferCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode the response structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

TransferCallRequest
This is used to build or decode a TransferCallRequest message to merge an held and an activeCall on this UA.

Method	Signature	Description
build	String build()	Encode the request structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_heldCall	Call	M	Call to retrieve and merge
_activeCall	Call	M	To merge with heldcall

TransferCallResponse
This is used to build or decode a TransferCallResponse, positive response.

Method	Signature	Description
build	String build()	Encode the response structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

Package com.m2msoft.uaCSTA.monitor

MonitorStartRequest
This is used to build or decode a MonitorStartRequest message to ask UA for events.

Method	Signature	Description
build	String build()	Encode the structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_monitorObject	String	M	URI of UA to monitor
_monitorType	String	M	Set to device. / Read Only at that time
requestedMonitorMediaClass	int	M	Bit mask for VOICE|IM. Voice only supported

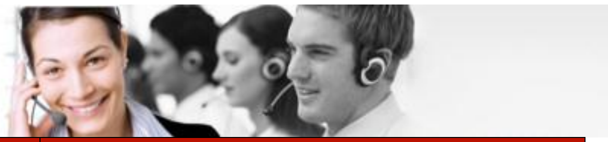
MonitorStartResponse
This is used to build or decode a MonitorStartResponse, positive response.

Method	Signature	Description
build	String build()	Encode a structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_monitorCrossRefId	String	M	Unique id to be found in all events sent from now

MonitorStopRequest
This is used to build or decode a MonitorStopRequest message to ask UA for events.

Method	Signature	Description
build	String build()	Encode the structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails



Attributes	Type	M/O	Description
_monitorCrossRefId	String	M	As given in the MonitorStartResponse from the UA
_monitorType	String	M	Set to device. / Read Only at that time
requestedMonitorMediaClass	int	M	Bit mask for VOICE|IM. Voice only supported

<u>MonitorStopResponse</u>
This is used to build or decode a MonitorStopResponse, positive response.

Method	Signature	Description
build	String build()	Encode a structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_none		M	

Package com.m2msoft.uaCSTA.events

This package contains events to send/receive. (only when a monitor has been requested on the UA)

<u>DeliveredEvent</u>
This is used to track alerting of calls.

Method	Signature	Description
build	String build()	Encode the structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_monitorCrossRefId	String	M	Always/see MonitorStartResponse
_originatedConnection	Call	O	Call IN PROGRESS
_callingDevice	String (deviceId)	O	
_calledDevice	Call	O	
_alertingDevice	String (deviceId)	O	Call RINGING
_establishedConnection	Call	O	Call CONNECTED
_answeringDevice	String (deviceId)		

EstablishedEvent
This is used to track connection of calls.

Method	Signature	Description
build	String build()	Encode the structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_monitorCrossRefId	String	M	Always/see MonitorStartResponse
_establishedConnection	Call	O	Call CONNECTED
_callingDevice	String (deviceId)	O	
_calledDevice	Call	O	
_answeringDevice	String (deviceId)	O	

ConnectionClearedEvent
This is used to track release of calls.

Method	Signature	Description
build	String build()	Encode the structured message into an uaCSTA XML string
decode	boolean decode(String data)	Decode an XML buffer in a structured object. Return false when decoding fails

Attributes	Type	M/O	Description
_monitorCrossRefId	String	M	Always/see MonitorStartResponse
_droppedConnection	Call	O	Call RELEASED
_callingDevice	String (deviceId)	O	
_calledDevice	Call	O	

7.2.18. Mini Call Center Development

With the S5000 and the GKXAPI, it is easy to build your own call center.
Let's see the main process.

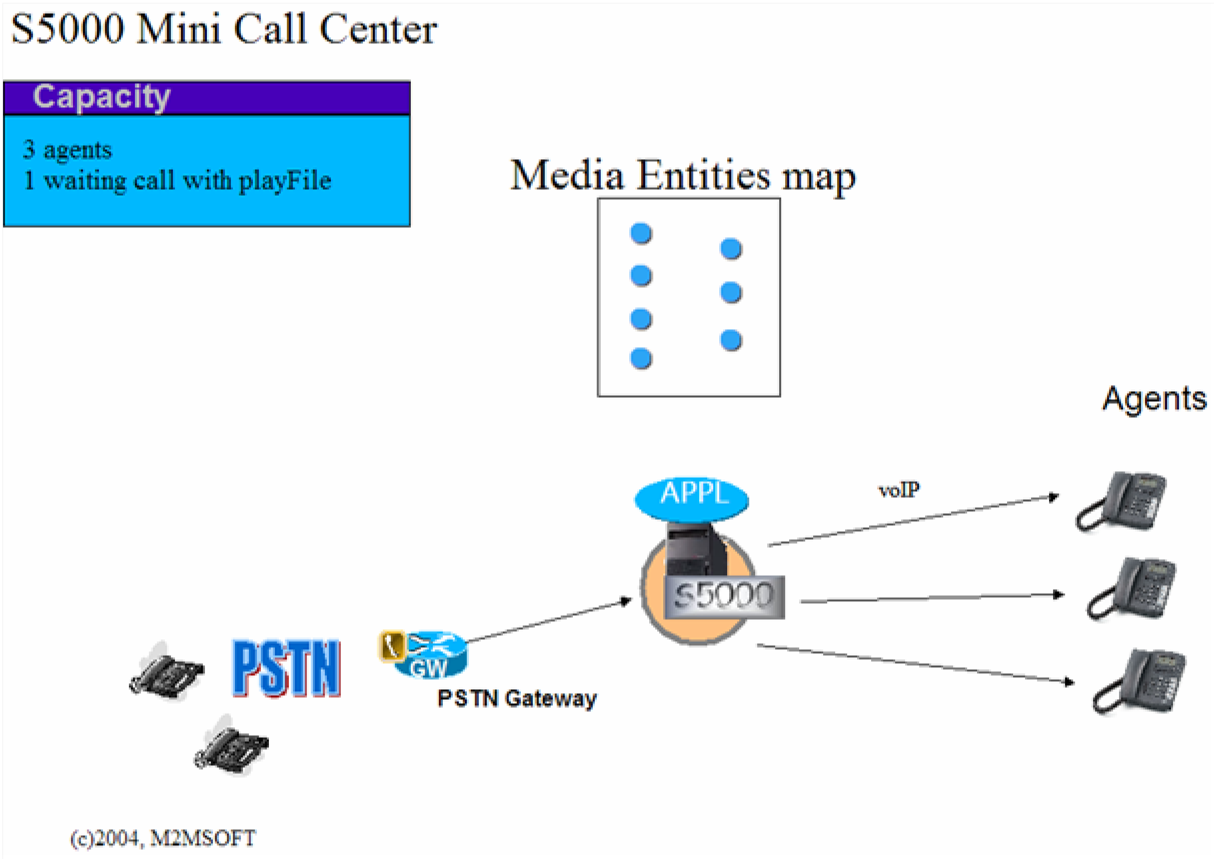


Fig.76 simple call center design

This figure depicts a small capacity call center with 3 agents (gathered in one waiting queue) and a call waiting queue of 1 call.

To make such an application, one must define first the number of Media Entity that we need. Here the number is 7. 3 to connect the agents simultaneously, 3 to connect the agents to their parties and 1 for a waiting call in queue.

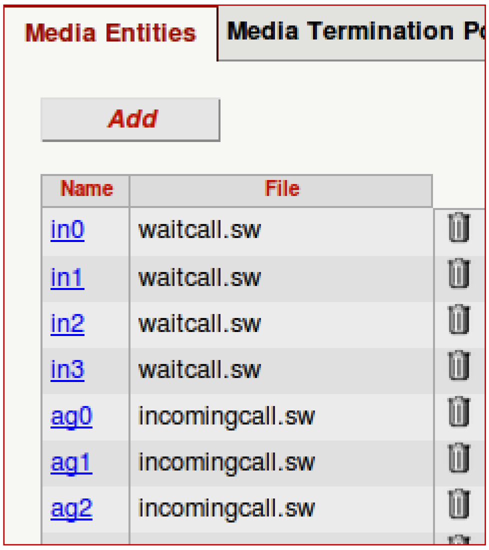
The pages below contain instructions and indications on such an application skeleton. This is not designed as a complete program but rather it shows how to implement a simple Automatic Call Distribution Automaton.

Step 1
Name these Media Entity as follow:
in1, in2, in3, in4 for the incoming calls Media Entity
ag1, ag2, ag3 for the agents.

Step 2
Define (and record) two announcements:
waitcall.sw for the customers
incomingcall.sw for the agents

Step3

Associate these announcements to the media entities in the S5000 configuration.



As an option one can define a supplementary Media Entity and a route (EmbeddedService) to it in case of unavailable channels on the call center.

Step 4

The application:

As we need to serve 4 simultaneous input calls and 3 simultaneous outgoing calls, we need 7 slots.
We need to define: 7 callIds and 7 state variables to track the MediaEntity pool availability.

```
String[] mediaIn=new String[4]; // mediaEntity names
String[] mediaAg= new String[3];
int[] mediaInState=new int[4];// mediaEntity states
int[] mediaAgState=new int[3];
String agE164 = new String[3];   // agents phone  numbers

String[] agCallIds = new String[3]; // store all agents calls

String currentAgCallId= » »;// current callref to join current searched agent
String currentWaitCallId= » »;// current customer callref. waiting in queue
int currentWaitME=-1;
int currentAgME=-1; //current MediaEntity used to contact agent

void initMedia()
{
 mediaIn[0]= »in0 »;
 mediaIn[1]= »in1 »;
 mediaIn[2]= »in2 »;
 mediaIn[3]= »in3 »;
 mediaAg[0]= »ag0 »;
 mediaAg[1]= »ag1 »;
 mediaAg[2]= »ag2 »;

 mediaInState[0]=0; // free
 mediaInState[1]=0; // free
 mediaInState[2]=0; // free
 mediaInState[3]=0; // free
```

```
 mediaAgState[0]=0; // free
 mediaAgState[1]=0; // free
 mediaAgState[2]=0; // free

 agE164[0]= »515 »;
 agE164[1]= »516 »;
 agE164[2]= »517 »;

 // no calls yet
 agCallIds[0]= » »;
 agCallIds[1]= » »;
 agCallIds[2]= » »;

}

public void processMessage(......)
{
   // Incoming call management
   case SETUP:
 // get available waiting mediaEntity
 int n=getFreeMediaIn();
 // check if n==-1 => no input channel, release call with setup_reply.reject()
 setup_reply.changeDestination(mediaIn[n]);
 currentWaitME=n; // save for later the mediaentity name
 currentWaitCallId=msg.callId;
 setup_reply.accept();
 mediaInState[n]=1;// mark as locked
 // get available agent
 int a=getFreeMediaAg();
 if (a>=0) {
  // there is an agent available, startCall to this agent number
  currentAgCallId=_api.getCallId();
  currentAgME=a;
  mediaAgState[a]=1; // locked agent
  agCallIds[a] = currentAgCallId; // save callid
  startCall(currentAgCallId, agE164[a], null, mediaAg[a]);
 }
 else {
  // we cannot call an agent now, wait in queue !
 }
 break;

   case OLCACK :
 // check if agent
 if (msg.callId.compareTo(currentAgCallId)==0) {
  // agent called connected now
  // join the calls
  _api.joinCall(mediaIn[currentWaitME], mediaAg[currentAgME]);
  currentAgME=-1; // no more agent search for now
 }
 break;

   case DISC:
 // check if an agent is released
 if (msg.callId.compareTo(agCallIds[0]==0) {
  // agent 0 is now free
  agCallIds[0]= » »;
  mediaAgState[0]=0; // agent is free
  // if we need to start call, call this agent now !
  if (currentAgME==-1 && currentWaitME!=-1) {
   // search is needed ! Start call here
   // there is an agent available, startCall to this agent number
   currentAgCallId=_api.getCallId();
```

```
    currentAgME=0;
    mediaAgState[0]=1; // locked agent
    agCallIds[0] = currentAgCallId; // save callid
    startCall(currentAgCallId, agE164[0], null, mediaAg[0]);
   }
  }
 else if (msg.callId.compareTo(agCallIds[1]==0) {
   // agent 1 is now free
   agCallIds[1]= » »;
   mediaAgState[1]=0; // agent is free
   // if we need to start call, call this agent now !
   if (currentAgME==-1 && currentWaitME!=-1) {
    // search is needed ! Start call here
    // there is an agent available, startCall to this agent number
    currentAgCallId=_api.getCallId();
    currentAgME=1;
    mediaAgState[1]=1; // locked agent
    agCallIds[1] = currentAgCallId; // save callid
    startCall(currentAgCallId, agE164[1], null, mediaAg[1]);
   }
  }
 else if (msg.callId.compareTo(agCallIds[2]==0) {
   // agent 2 is now free
   agCallIds[2]= » »;
   mediaAgState[2]=0; // agent is free
   // if we need to start call, call this agent now !
   if (currentAgME==-1 && currentWaitME!=-1) {
    // search is needed ! Start call here
    // there is an agent available, startCall to this agent number
    currentAgCallId=_api.getCallId();
    currentAgME=2;
    mediaAgState[2]=1; // locked agent
    agCallIds[2] = currentAgCallId; // save callid
    startCall(currentAgCallId, agE164[2], null, mediaAg[2]);
   }
  }
 else {
   // currentWaitCallId : customer waiting has released
   if (msg.callId.compareTo(currentWaitCallId)==0) {
    // waiting queue is now free
    currentWaitCallId= » »;
    mediaInState[currentWaitME]=0; // in free
    currentWaitME=-1;
   }
  }
   break;
}
```

7.3. C API (GKXAPI)

7.3.1. How does it work?

The programmer needs the following files to work with:
libgkxapi.a
gkx.h

This contains all necessary functions and defines to develop and run an application towards M2M-S5000.

7.3.2. My HELLO WORD

This chapter details a complete call management application and makes use of the functions and values changes.

Due to the evolving nature of the API, we cannot guarantee that the code depicted here is exactly working with your API version; but we deliver up to date source sample with all our API packages.
Ask your M2MSOFT representative for the working code sample.

Application specifications

My Hello World is a sample application that perform the following:

- Connect to S5000 and initialize 5 contexts.
- Display received event messages
  - Case RRQ
    E164_Source=1001 → Reject
    E164_Source=1002 → Change to 1003 + 1004 and Accept
    else → Accept
  - Case ARQ (originated from caller)
    E164_Destination=2001 → Reject
    E164_Destination=2002 → Change to 2003 and Accept
    else → Accept
  - Case ARQ (Originated from called) → Always accept
  - Case SETUP
    E164_Destination=4001 → Reject
    E164_Destination=4002
      Change Destination to 4003
      Change Display for « coucou »
      Private Data field = « myDatas »
      Accept
    else → Accept
  - Case CONNECT → Console display « Connection CallID <callId> »
  - Case DATA
    When PrivateData equals « PleaseURQ » → Send GK an URQ for IP=1.1.1.1
    When PrivateData equals « PleaseRelease » → Send GK RELEASE of the call whose CallId=abcd0000ffff

Step by step programming

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "gkx.h"

#define VERSION "1.5"

int process(GKMSG *, CTX *);

int main(int argc, char *argv[])
{
  char  IPADD[50]="10.162.56.151";
  int   PORT=16000;
  int   i;
  bool traceMsg=false;

  printf("Client test API - Version %s\nGKXAPI - Version
%s\n\n",VERSION,GKXAPI_VERSION);
  for (i=1;i<argc;i++)  {
    if (strcmp(argv[i],"-g")==0)  {
      if (argc-1<++i) {
        i=-1;
        break;
      }
      strcpy(IPADD,argv[i]);
    }
    else if (strcmp(argv[i],"-p")==0) {
      if (argc-1<++i) {
        i=-1;
        break;
      }
      PORT=atoi(argv[i]);
    }
    else if (strcmp(argv[i],"-t")==0) traceMsg=true;
    else  {
      i=-1;
      break;
    }
  }
  if (i==-1)  {
    printf( "Syntax : %s [options]\n\n"
            "Options :\n"
            "  -g <gatekeeper (or host) ip address>\n"
            "  -p <port>\n"
     "  -t (for API messages trace)\n\n",argv[0]);
    exit(1);
  }

  GKX *gkx=new GKX(traceMsg);
  if (gkx->Open(IPADD,PORT,5)==-1)  {
    printf("Failed to connect to %s:%d\n",IPADD,PORT);
    return 0;
  }
  gkx->start(5);
  gkx->addCallBack(process);
  gkx->ReadWriteDatas();
  //gkx->Close();
  return 0;
}
```

```
int process(GKMSG *gkMsg, CTX *ctx)
{
  char  strMsgType[20];

  switch(gkMsg->type) {
    case MSG_RRQ:     strcpy(strMsgType,"MSG_RRQ");     break;
    case MSG_URQ:     strcpy(strMsgType,"MSG_URQ");     break;
    case MSG_ARQ:     strcpy(strMsgType,"MSG_ARQ");     break;
    case MSG_aRQ:     strcpy(strMsgType,"MSG_aRQ");     break;
    case MSG_DRQ:     strcpy(strMsgType,"MSG_DRQ");     break;
    case MSG_LRQ:     strcpy(strMsgType,"MSG_LRQ");     break;
    case MSG_RAI:     strcpy(strMsgType,"MSG_RAI");     break;
    case MSG_SETUP:   strcpy(strMsgType,"MSG_SETUP");   break;
    case MSG_CONNECT: strcpy(strMsgType,"MSG_CONNECT"); break;
    case MSG_BUSY:    strcpy(strMsgType,"MSG_BUSY");    break;
    case MSG_DISC:    strcpy(strMsgType,"MSG_DISC");    break;
    case MSG_DATA:    strcpy(strMsgType,"MSG_DATA");    break;
    case MSG_UNKNOWN: strcpy(strMsgType,"MSG_UNKNOWN"); break;
  }

  printf( "type=%s\n"
          "transaction=%s\n"
          "callId=%s\n"
   "sessionId=%s\n"
          "ip_source=%s\n"
          "e164_source=%s\n"
          "h323_source=%s\n"
          "ip_destination=%s\n"
          "e164_destination=%s\n"
          "h323_destination=%s\n"
   "display=%s\n"
          "techPrefix=%s\n"
          "bandwidth=%s\n"
          "outOfResource=%d\n"
   "gateway=%d\n"
          "data=%s\n\n",
          strMsgType,
          gkMsg->transaction,
          gkMsg->callId,
   gkMsg->sessionId,
          gkMsg->ip_source,
          gkMsg->e164_source,
          gkMsg->h323_source,
          gkMsg->ip_destination,
          gkMsg->e164_destination,
          gkMsg->h323_destination,
   gkMsg->display,
          gkMsg->techPrefix,
          gkMsg->bandwidth,
          gkMsg->outOfResource,
   gkMsg->gateway,
          gkMsg->data);

  RRQ_REPLY   *RRQ=new RRQ_REPLY(gkMsg, ctx);
  ARQ_REPLY   *ARQ=new ARQ_REPLY(gkMsg, ctx);
  aRQ_REPLY   *aRQ=new aRQ_REPLY(gkMsg, ctx);
  SETUP_REPLY *setup=new SETUP_REPLY(gkMsg, ctx);

  switch(gkMsg->type)  {
    case MSG_DATA:
    if (strcmp(gkMsg->data,"PleaseURQ")==0) ctx->sendURQ("1.1.1.1");
    else if (strcmp(gkMsg->data,"PleaseRelease")==0)
        ctx->releaseCall("abcd0000ffff");
    break;
```

```
    case MSG_RRQ:
    if (strcmp(gkMsg->e164_source,"1001")==0)  RRQ->reject();
    else if (strcmp(gkMsg->e164_source,"1002")==0)  {
      RRQ->changeE164("1003,1004");
      RRQ->accept();
    }
    else  RRQ->accept();
    break;

    case MSG_ARQ:
    if (strcmp(gkMsg->e164_destination,"2001")==0)  ARQ->reject();
    else if (strcmp(gkMsg->e164_destination,"2002")==0)  {
      ARQ->changeE164Destination("2003");
      ARQ->reject();
    }
    else  ARQ->accept();
    break;

    case MSG_aRQ:
    aRQ->accept();
    break;

    case MSG_SETUP:
    if (strcmp(gkMsg->e164_destination,"4001")==0)  setup->reject();
    else if (strcmp(gkMsg->e164_destination,"4002")==0)  {
      setup->changeE164Destination("4003");
      setup->changeDisplay("coucou");
      setup->setPrivateData("myDatas");
      setup->accept();
    }
    else  setup->accept();
    break;

    case MSG_CONNECT:
    printf("Connection callId %s\n",gkMsg->callId);
    break;
  }

  return 1;
}
```

7.3.3. Classes

The classes used within GKXAPI are:

Class / Interface	Description
GKX	The main class used to connect to M2M-S5000 and arm callback management.
GKMSG	The event structure passed to the user function. The user can access and modify a number of attributes issued from the H323 or SIP message.
CTX	As a multi-threaded application, a context specific object is handled to the user method as well. **Used to store data and to send commands for the call (Release)**
RRQ_REPLY	Used to activate any change, modifiy, accept and reject of H323 RRQ or SIP REGISTER event
ARQ_REPLY	Used to activate any change, modifiy, accept and reject of initial ARQ event
aRQ_REPLY	Used to activate any change, modifiy, accept and reject of final ARQ event
SETUP_REPLY	Used to activate any change, modifiy, accept and reject of SETUP or SIP INVITE event
CONNECT_REPLY	Used to accept or reject a CONNECT event. Some fields can be set at that time, as the display value.

7.3.4. Class GKX

This class manages the connection with the S5000 and the global internal scheduling of events and callbacks. There are general functions and half calls management functions.

Method	Signature	Description
GKX	GKX()	Constructor for main object GKX
Open	int Open(char *ip, int port, int timer)	Open a connection to a M2M-S5000 at the *ip* address and TCP *port*. The *timer* parameter is the timeout TCP connection. Return 0=success, -1=failed
Close	void Close()	Close connection to 5000.
ReadWriteDatas	void ReadWriteDatas()	Main loop process
addCallBack	void addCallBack(process) *callback function:* *void process (GKMSG *, CTX *)*	Provide the process() method as callback function to receive events.
start	void start(int nbSlots)	Initiate a number of contexts with GK.
	void start(int nbSlots, char * appName)	Initiate contexts and application name.
setForcedRoute	void setForcedRoute(bool forced)	When *forced* is TRUE all the call are forwarded to this application, no embeddedService is needed.
addListen	int addListen(int pkg)	Add an event set to be handled within the application. Standard values are : - H245 - RSVP - INFO-CSTA - SUPSERV (H450 events) This adds new events to be handled according to the underlying protocol.
getVersion	char *getVersion()	Return the GKXAPI version.

Attributes	Type	Description
none		

7.3.5. Class GKMSG

This class is related to a M2M-S5000 event and stores a set of attributes related to the event.
Event can be: (at time of writing)

- **Default event package : (SIP and H323)**
  - RRQ, URQ, ARQ, SETUP, ALERTING, CONNECT, DISC and TIMEOUT.
  - INFO, INFOOK, RCF, RRJ, RTIMEOUT
  - EPLIST
  -
- **H245 event package:**
  - OLC, OLCACK

- **RSVP event package:**
  - RSVP_PATH, RSVP_RESV, RSVP_RESVCONF, RSVP_PATHTEAR

- **MediaEntities event package**
  - ME_MESPLITTED, ME_MECREATED, ME_MECREATERR

- **Supplementary services event package (call transfer, etc)**
  - SUPSERV (for H450 events)

Signaling events

SIP message	H323 message	H245 message	RSVP msg	GKX event MSG_
REGISTER	RRQ			RRQ
INVITE	SETUP			SETUP
BYE, CANCEL	ReleaseComplete			DISC
ACK (contextual)	CONNECT			CONNECT
Ringing	ALERTING			ALERTING
		OpenLogicalChannel		OLC
		OpenLogicalChannelAck		OLCACK
			Path	RSVP_PATH
			Resv	RSVP_RESV
			ResvConf	RSVP_RESVCONF
			PathTear	RSVP_PATHTEAR
			ResvErr	RSVP_RESVERR
			PathErr	RSVP_PATHERR
INFO				INFO
OK on INFO				INFOOK
OK on REGISTER				RCF
Ko on REGISTER				RRJ
Timeout on register				RTIMEOUT
REGISTER with 0 TTL	URQ			URQ
Endpoint unreachable	Endpoint unreachable			URQ
	H450			SUPSERV
				LOSTCNX (lost S5000 link)

Response events

Response events are sent in response to commands.

- Media Entity objects accept commands and the table below shows the awaited events;
- Generic commands expects events in return, as getEPList(). Details follow.

Media Entity command	GKX event	Comment
Creation success Creation error	ME_MECREATED ME_MECREATERR	
Split success Split error	ME_MESPLITTED ME_MESPLITERR	

General Command	GKX event MSG_	Comment
Endpoint list request getEPList()	EPLIST -infoData field contains the endpoint list in an <xml> like view -aliasList field contains the list of all SIP and H323 endpoint aliases, each-one separated by a coma	The endpoint list is given as a data string that contains the database description as below, in pseudo BNF: **EPList:** entrylist END **entrylist:** entry SEP entrylist | entry **entry:** <class=classval ; type=typeval ; alias=aliasval ; contactAliases=contactAliasesVal;ttl=ttlval;info=infoval > **classval:** H323 | SIP **typeval:** integer (reserved) **contactAliasesVal:** String,String,… **aliasval:** string **ttlval:** integer **infoval:** string SEP: \n END: \n\n

End points list EPLIST infoData field format (char *)

Entry Attribute	**class** (String)	**type** (integer)	**aliases** (String)	**ttl** (integer)	**info** (String)
Description	H323 or SIP	For future use	Endpoint known aliases, E164, H323-ID, URI; coma separated values	Time to live as known (may change)	Product and vendor information as extracted from the endpoint messages

```
Example :
<class=H323;type=1;aliases=60005,SIEMNS;ttl=15;info="Hinet LP5100">\n
<class=SIP;type=0;alias=5100@192.168.0.30:5060,Office;contactAliases=5101;5
102;ttl=2;info="Swissvoice IP10S">\n
```
\n

Method	Signature	Description
None		This object is passed in 1st parameter in callback function. No constructor is needed.

Attributes	Type	Description
type	enum _msgType	Message type (see below)
transaction	Char[10]	Transaction ID
callId	Char[40]	Call ID
sessionId	char[10]	Session ID
ip_source	char[30]	IP source address
e164_source	char[128]	E164 source address
h323_source	char[128]	H323 source address
ip_destination	char[30]	IP destination address
e164_ destination	char[128]	E164 destination address
h323_ destination	char[128]	H323 destination address
techPrefix	char[128]	List of gateways'TechPrefix
bandwidth	char[30]	Bandwidth
outOfResource	Bool	E1 gateway out of resource flag
gateway	Bool	EndPoint is gateway flag
display	char[128]	Display field
data	char[128]	Data field for GK to client communication
privateData	char[128]	Data field within Setup/Invite messages

_msgType	Reply required / Notification only
MSG_RRQ	Reply required
MSG_URQ	Notification only
MSG_ARQ	Reply required
MSG_aRQ	Reply required
MSG_DRQ	Notification only
MSG_LRQ (*)	Reply required
MSG_LCF (*)	Notification only
MSG_RAI (*)	Notification only
MSG_SETUP	Reply required
MSG_CONNECT	Notification only
MSG_BUSY	Notification only
MSG_DISC	Notification only
MSG_DATA	Notification only
MSG_EPLIST	Notification only
MSG_UNKNOWN	Notification only

Messages to fields mapping

	RRQ	URQ	ARQ	aRQ	DRQ	RAI(*)	SETUP	CONNECT	DISC	DATA
type	X	X	X	X	X	X	X	X	X	X
transaction	X		X	X			X	X	X	
callId			X	X	X		X	X	X	
sessionId							X	X	X	
ip_source	X	X	X	X		X	X			
e164_source	X	X	X	X			X			
h323_source	X	X	X	X			X			
ip_destination			X	X			X			
e164_ destination			X	X			X			
h323_ destination			X	X			X			
techPrefix	X									
bandwidth			X	X						
outOfResource						X				
gateway							X			
display							X	X		
data										X
privateData							X			

7.3.6. Class CTX

The CTX object can be used to act on any call handled within the application:

- release a call, place a call, join a call, …

Method	Signature	Description
releaseCall	void releaseCall(char *callID)	Disconnect call or half call.
sendURQ	void sendURQ(char *ip)	Send URQ command from GK to *ip* address of endpoint.
getCallId	char *getCallId()	Allocate a callIdentifier string. This string is to use for subsequent startCall(). Freeing of the returned string is to perform by user.
startCall	int startCall(char *callId, char *e164Called, char *destIpAddr, char *e164Calling, char *display)	Uses a MediaEntioty to dial out an outgoing call to e164Called or destIpAddress (GW case). E164Calling is a valid and unconnected mediaEntity within the S5K gatekeeper. Return 0=success, -1=failed.
joinCall	int joinCall(char *media1, char *media2)	Join 2 halves communications. Media1 and Media2 are two connected mediaEntities. This can be the next step of trwo startCall(). Join halves coms stops any play file that was playing and connects the media channels in both ½ calls. Return 0=success, -1=failed.

Attributes	Type	Description
none		

7.3.7. Class RRQ_REPLY

This class is used to build a RRQ answer that M2M-S5000 will use for its actions.
A RRQ_REPLY object can only be built while in a RRQ event processing.
The application can choose to reject the endpoint registration request at that point with a reject() or to continue the registration with an accept().
When the decision is to continue the registration, one can set/modify the endpoint elements with changeE164() method for example to force a set of dynamic aliases.
NOTE: A RRQ event is thrown for H323-RRQ or SIP-REGISTER messages.

Method	Signature	Description
RRQ_REPLY	RRQ_REPLY (GKMSG *gkMsg, CTX *ctx)	Constructor for response to initial ARQ event. (gkMsg and ctx passed within callback).
changeE164	void changeE164(char *newE164)	Change the fist E164 alias found with this one. The endpoint will act as if this dynamic alias had been set in first place.
changeH323	void changeH323Id(char *newH323Id)	Change the fist H323Id alias found with this one. The endpoint will act as if this dynamic alias had been set in first place.
accept	void accept()	Send the response to S5000 and accept the request.
reject	void reject()	Send the response to S5000 and reject the request.

Attributes	Type	Description
none		

7.3.8. Class ARQ_REPLY

This class is used to build an INITIAL ARQ answer that M2M-S5000 will use for its actions.
An ARQ_REPLY object can only be built while in a ARQ event processing.

Method	Signature	Description
ARQ_REPLY	ARQ_REPLY (GKMSG *gkMsg, CTX *ctx)	Constructor for response to initial ARQ event. (gkMsg and ctx passed within callback).
changeIpSource	void changeIpSource(char *newIP)	Change ip source in reply
changeE164Source	void changeE164Source(char *newE164)	Change e164 source in reply.
changeH323Source	void changeH323Source(char *newH323)	Change h323Id source in reply.
changeIpDestination	void changeIpDestination(char *newIP)	Change ip destination in reply.
changeE164Destination	void changeE164Destination(char *newE164)	Change e164 destination in reply.
changeH323Destination	void changeH323Destination(char *newH323)	Change h323 destination in reply.
changeBandwidth	changeBandwidth(char *newBW)	Change bandwidth in reply with string format.
accept	accept()	Send the response to S5000 and accept the request.
reject	reject()	Send the response to S5000 and reject the request.

Attributes	Type	Description
none		

7.3.9. Class aRQ_REPLY

This class is used to build an FINAL ARQ answer that M2M-S5000 will use for its actions.
An aRQ_REPLY object can only be built while in a aRQ event processing.

Method	Signature	Description
aRQ_REPLY	ARQ_REPLY (GKMSG *gkMsg, CTX *ctx)	Constructor for response to final ARQ event. (gkMsg and ctx passed within callback).
changeIpSource	void changeIpSource(char *newIP)	Change ip source in reply
changeE164Source	void changeE164Source(char *newE164)	Change e164 source in reply.
changeH323Source	void changeH323Source(char *newH323)	Change h323Id source in reply.
changeIpDestination	void changeIpDestination(char *newIP)	Change ip destination in reply.
changeE164Destination	void changeE164Destination(char *newE164)	Change e164 destination in reply.
changeH323Destination	void changeH323Destination(char *newH323)	Change h323 destination in reply.
changeBandwidth	changeBandwidth(char *newBW)	Change bandwidth in reply with string format.
accept	accept()	Send the response to S5000 and accept the request.
reject	reject()	Send the response to S5000 and reject the request.

Attributes	Type	Description
none		

7.3.10. Class SETUP_REPLY

This class is used to build a SETUP answer that M2M-S5000 will use for its actions.
A SETUP_REPLY object can only be built while in a SETUP event processing.
The SETUP_REPLY object contains all the information for the resulting (and hence modified) SETUP message M2M-S5000 might forward to the other party.
A SETUP_REPLY object contains all the initial SETUP message values and the developer can modify some of these via object methods. For example, one can modify the display element with changeDisplay() method.

NOTE: a SETUP event is thrown for H323-SETUP or SIP-INVITE messages.

Method	Signature	Description
SETUP_REPLY	SETUP_REPLY (GKMSG *gkMsg, CTX *ctx)	Constructor for response to SETUP event. (gkMsg and ctx passed within callback).
changeE164Source	void changeE164Source(char *newE164)	Change e164 source in reply.
changeIpDestination	void changeIpDestination(char *newIP)	Change ip destination in reply.
changeE164Destination	void changeE164Destination(char *newE164)	Change e164 destination in reply.
changeDisplay	void changeDisplay(char *newDisplay)	Change display in reply.
setPrivateData	setPrivateData(char *data)	Insert private data string (max 80) in Setup/Invite message.
setNoAutoconnect	setNoAutoconnect()	Transform a normal call into a ½ call. Redirect the incoming call to a mediaEntity and use this functions to avoid connecting automatically the call to the playFile. SetNoAutoconnect() let the call rings.
accept	accept()	Send the response to S5000 and accept the request.
reject	reject()	Send the response to S5000 and reject the request.

Attributes	Type	Description
none		

7.3.11. Class CONNECT_REPLY

This class is used to build a CONNECT answer that M2M-S5000 will use for its actions.
A CONNECT_REPLY object can only be built while in a CONNECT event processing.
The application can choose to stop the call at that point with a reject() - a ReleaseComplete H323 will be generated in the H323 call legs- or to continue the call with an accept().
When the decision is to continue the call, one can set/modify the display element with changeDisplay() method in order to display useful information to the caller.

Method	Signature	Description
CONNECT_REPLY	CONNECT_REPLY (GKMSG *gkMsg, CTX *ctx)	Constructor for response to SETUP event. (gkMsg and ctx passed within callback).
changeDisplay	void changeDisplay(char *newDisplay)	Change display in reply.
setJoinCallId	setJoinCallId(char *callId)	Connects two halves communications. The waiting call callId (peer call) is given to the function and as the current call connects, it automatically will have its media channels connected to the peer call.
accept	accept()	Send the response to S5000 and accept the request.
reject	reject()	Send the response to S5000 and reject the request.

Attributes	Type	Description
none		

8. Administration via TCP Socket API

8.1. Endpoints

S5000 has capabilities to manage provisioning of different IP-Phones.This process uses API socket (default port: 16000) for operate provisioning of phones like Thomson (ST2030, TB30), Panasonic (TGP550, UT136) and Aastra (6757i and 6731i).

Three operations are available; Creation/Edition, Suppression and Consultation of different profiles.

With ***PROVEP***, command we can create new profile, but it's possible to suppress, consult or modify existing profile too. TCP protocol transport is used to ensure security in transactions

```
PROVEP s :1 e:510001 t:Aastra-6731i m:001124420AC74 D:Thom_5122 n:2 z:-1
u:-1 H:1 f:None ft:10 fm:0 $:1 gr:m2m li:5122 p:5122 v:0 vo:0 vd:100 P:20
FONCT:3;BLF;5110
```

This command creates new profile of 510001 with type Aastra-6731i.

Attributes

To create new endpoint profile with auto provisioning, it's essential to fill operation (s :) and alias (e :). Any PROVEP command successes without these parameters. They permit to create configuration files to move on the phone by s5000.

Different attributes are represented in PROVEP command:

Attribute	Definition
s:	Represente the type of operation to affect to the command, **0:Suppression 1 :Création/Modification 2 : Consultation**
e:	Represente the alias of the IP-phone
t:	Represente the type of ip-phone. It takes value in {Thomson-ST2030, Thomson-TB30, Panasonic-TGP550, Panasonic-UT136, aastra-6731i, Aastra-6757i, none}.
m:	Is the mac Address
n:	Is the number of lines, for the Aastra this field is unused
z:	Represente the Time Zone. Look at Time Zone table for supported value.
u:	This field represente selected language of phone, default langage is selected while not filled.
c:	county tone , depending of the kind of phone different tables are present
H:	DHCP, 0=false 1=true
@:	static IP
S:	The mask
G:	Gateway
f:	Forward, it's value in {**None, Always, Busy**}

fn:	Forward number
ft:	Forward timer by default value is 10,
fm:	Forward message this field can take value in {**0,1**}
$:	Ethernet connexion **0 :** Auto, **1 :** 100/Half , **2 :** 100/Full **3 :** 10/Half**, 4 :**10/Full
gr:	Represente the group of the phone
R:	Represente the restriction list
li:	The login of sipAccount
p:	The password of sipAccount
v:	Represente vlan field 0=false 1=true
vl:	Vlan Voice, allowed only if vlan enabled (**v :1)**
vd:	Vlan data, depends on vlan field
P:	Represente the Ptime in case of G729 codec
B:	Represente supervised line 0=false et 1=true
C:	Represente list of shared lines. Elements of list are separed by coma “,”
O:	Represente “Call Waiting Tone disabled” field 0=false et 1=true
E:	Represente “Distinguished melody for external calls” field values ares 0=false et 1 = true by default value is 0
y:	Represente « Préfix Sip Appelant pour Transfert » field.
l:	Represente web password.
r:	Represente chosen melody. Taken value are represented in table of Melody
FONCT:	Represent list of functionals touches**.** Different fonctionals touches are separed by “>” Touches are represented by ***Value, Function, Number*** and in case of Aasra-6757i**, *Label***. The label is the name of the touch represented in the sceen. All fields of touch are separed by semicolon “**;**” ✓ ***Value*****:** represente le position of the touch ✓ ***Function*****:** values of this field are: **BLF** **SPEEDDIAL** **DTMF (**not for Aastra). ✓ ***Number*****:** represente an endpoint number ✓ ***Label:*** only for Aastra_6757i Ex: `FONCT: 4;5170;DG>5;5425;DT`

Endpoint Profile without Auto-provisioning (PROVEP s: 1)

There are fields whose presence is essential in the application to create a profile with auto-provisioning.
Indeed you can't provision a phone which you don't know the type (None). In this case you create a profile of the phone without auto-provisioning.

Request:

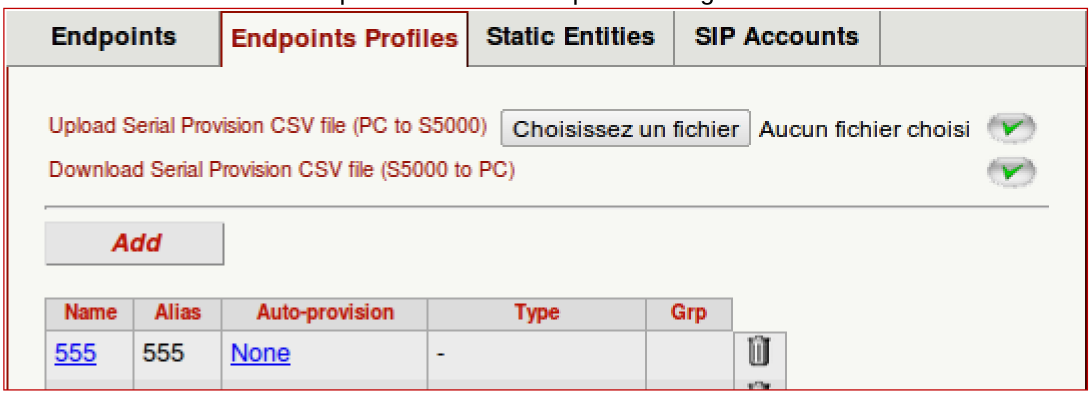
```
PROVEP s:1 e:555 t:None m:001124420AC741 D:555 n:2 z:-1 u:-1 H:1 f:None
ft:10 fm:0 $:1 gr:m2m li:5122 p:5122 v:0 vo:0 vd:100 P:40
```

Response:

```
PROVEP T: 1
#:<class=PROV;name=555;operation=Creation;result=Succes;comment=No_PROVISIO
NING>
```

`<Comment=No_PROVISIONING>:` is precision on the provisioning operation, although the profile creation was success `<result=Success>`, but `<comment=No_PROVISIONING>` indicate that phone type chosen don't allow auto-provisioning "t: None".

In web interface we see 555 profile without auto-provisioning.



We obtain same result with the request

```
PROVEP s:1 e:555 t:None D:555 n:2 z:-1 u:-1 H:1 f:None ft:10 fm:0 $:1
gr:m2m li:5122 p:5122 v:0 vo:0 vd:100 P:40
```

Here t: None is chosen to for creating profile without auto-provisioning. That's why any macAddress was defined.

Comments

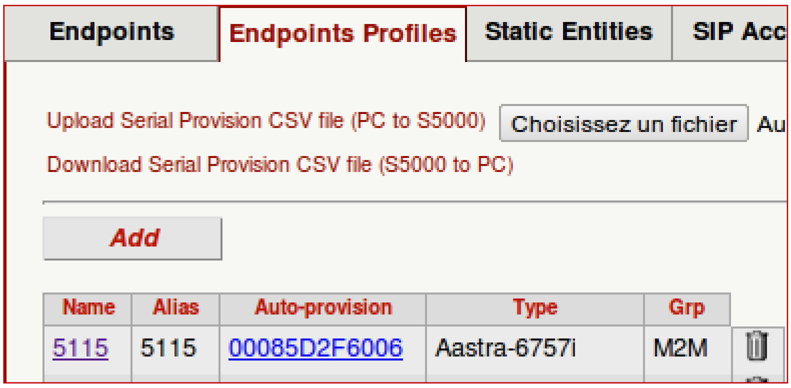
Responses of several requests contain comment which permit to precise the reason of failure of auto-provisioning.
In case of success any comment is provided.
These comments are:

- ✓ **NO_MACADDRESS :** indicate that macAddress is missed
- ✓ **NO_PROVISIONING** : provisioning is not pssible with current request
- ✓ **NO_PHONETYPE** : specify valable type of phone for auto-provisioning
- ✓ **NO_IPADDRESS :** this comment indicate that your DHCP is disabled but no IP address is specified
- ✓ **NO_ALIAS :** reqtest is unvailable , specify alias et retry
- ✓ **BAD_SYNTAXE :** verify sysntaxe and retry

Endpoint Profile with Auto-provisioning (PROVEP s: 1)

To create profile with auto-provisioning, we must inform phoneType with other value than"None", and mac Adress of the phone.

```
PROVEP s:1 e:5115  t:Aastra-5767i m:00085D2F6006 D:5115 n:2 z:-1 u:-1 H:1
f:None ft:10 fm:0 $:1 gr:m2m li:5122 p:5122 v:0 vo:0 vd:100 P:40
PROVEP T:1 #:<class=PROV;alias=5115;operation=Creation;result=Success>
```



Success of creation, new profile with provisioning is materialized by alias, macAddress and type in web interface.

To control all parameters entered click in macAdress of the phone in web interface

Functionals Touches (FONCT :)

The management of functional keys depends of the type of phone to manipulate.

- **Thomson's**

In Thomson phones as ST2030 or TB30, number of functional keys depends on line number. When line number is ten (n:10), any functional key can be specified.

DHCP	☑
IP addr (if static)	
Mask (if static)	
Gateway (if static)	
Ethernet connection	Auto
SIP Transport	UDP
VLANs	☐ Voice: 2 - Data: 1
Number of lines	10
Sip authentication login	
Sip authentication password	
Display	
Timezone	GMT
Language	Default
Country Tone	Default
Melody	Default
Distinguished melody for external calls	☐
Call Waiting Tone disabled	☐
F1	Line
F2	Line
F3	Line
F4	Line
F5	Line
F6	Line
F7	Line
F8	Line
F9	Line
F10	Line

Submit Cancel

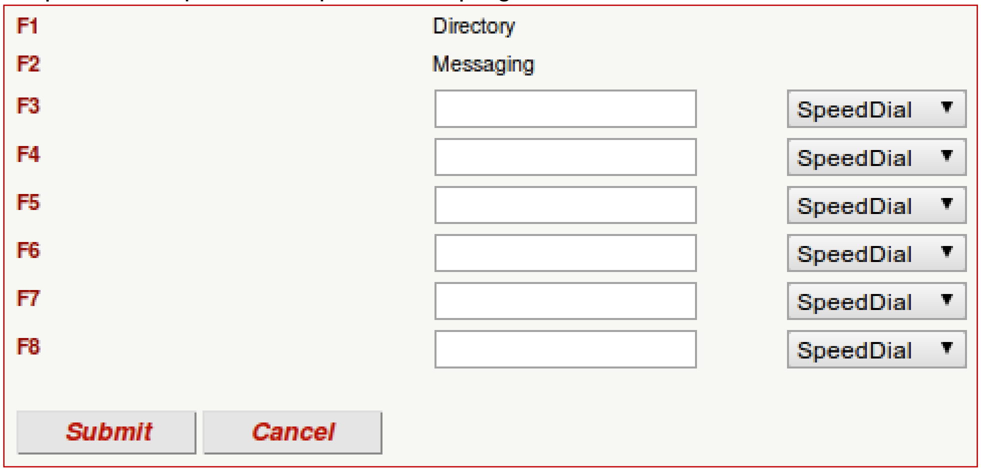
In Thomson's phone ***if n : k with k<10 then FONCT : i ;SERVICE ; NUM exists only if i element of interval [k+1,10], and SERVICE in {BLF, SPEEDDIAL, DTMF}***

Ex: `PROVEP s:1 e:444 t:Thomson-ST2030 m:0014758A1487 n:2 FONCT:3;BLF;5112`

F1	Line	
F2	Line	
F3	5112	Supervision
F4		SpeedDial

➢ **Aastra-6731i**

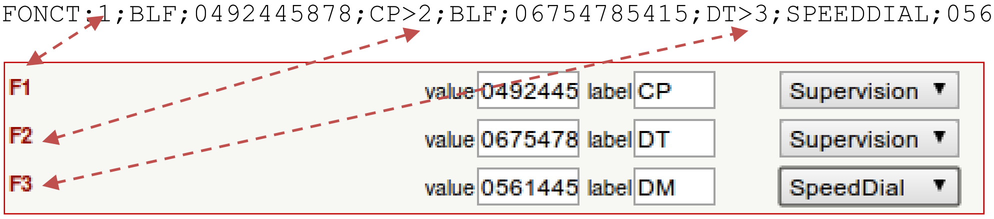
Aastra 6731i contains 8 functional keys the both first are used to specify (Directory, Messaging), at the 3th position it's possible to put the first programmable touch.



➢ **Aastra-6757i**

For Aastra 6757i, there have 6 functional touches named **softkeys.**
The softkeys have label field. Label is screened on the phone interface.
It's possible to customize label

```
Ex  PROVEP s:1 e:0624556677 t:Aastra-6757i m:003324420AC74
D:Aastra_Commercial n:2 z:-1 u:-1 H:1 f:None ft:10 fm:0 $:1 gr:m2m li:5122
p:5122 v:0 vo:0 vd:100 P:20
FONCT:1;BLF;0492445878;CP>2;BLF;06754785415;DT>3;SPEEDDIAL;0561445120;DM
```



Modification (PROVEP s: 1)

With the command PROVEP, edit a profile is to delete and recreate it with new values. The removing step is transparent to the user.

Consultation (PROVEP s: 2)

To ensure the creation of a profile with or without provisioning, you can make a consultation. The command "PROVEP s: 2" to query the table Endpoints profiles in consultation
PROVEP takes as input the transaction (s 2) and the alias of the Endpoint (e 555), and returns all the information about it.
To view information on all Endpoints table, we put "*" (PROVEP s: 2 e:*)

Request

```
PROVEP s:2 e:*
```

Response

```
PROVEP T:1 #:<class=PROV;alias=59889;mac=001124420AC74;phoneType=Aastra-
6731i;display=tompouce;restric=;sharedLines=;webPass=;PrefixTrans=;melody=0
```

```
;Dhcp=1;ip=;mask=;gateway=;login=5122;password=5122;ethCX=1;nbLines=2;cTone
s=;timeZone=15;langue=6;cWToneDisabled=0;supervisedCall=0;dustinguishMforEx
ternal=0;forwardType=None;forwardNum=;forwardMsg=0;group=m2m;F1=BLF+5110;F2
=;F3=;F4=;F5=;F6=;F7=;F8=>
<class=PROV;alias=5989;mac=001124420AC73;phoneType=Aastra-
6731i;display=tompouce;restric=;sharedLines=;webPass=;PrefixTrans=;melody=0
;Dhcp=1;ip=;mask=;gateway=;login=5122;password=5122;ethCX=1;nbLines=5;cTone
s=;timeZone=7;langue=1;cWToneDisabled=0;supervisedCall=0;dustinguishMforExt
ernal=0;forwardType=None;forwardNum=;forwardMsg=0;group=m2m;F1=BLF+5110;F2=
SPEEDDIAL+5214;F3=;F4=;F5=;F6=;F7=;F8=>
<class=PROV;alias=444;mac=;phoneType=None;display=;restric=;sharedLines=;we
bPass=;PrefixTrans=;melody=0;Dhcp=1;ip=;mask=;gateway=;login=;password=;eth
CX=1;nbLines=1;cTones=;timeZone=-1;langue=-
1;cWToneDisabled=0;supervisedCall=0;dustinguishMforExternal=0;forwardType=;
forwardNum=;forwardMsg=0;group=>
<class=PROV;alias=8900;mac=4545140021;phoneType=Aastra-
6757i;display=;restric=;sharedLines=;webPass=;PrefixTrans=;melody=0;Dhcp=1;
ip=;mask=;gateway=;login=;password=;ethCX=1;nbLines=1;cTones=;timeZone=-
1;langue=-
1;cWToneDisabled=0;supervisedCall=0;dustinguishMforExternal=0;forwardType=;
forwardNum=;forwardMsg=0;group=;F1=;F2=;F3=;F4=;F5=;F6=>
<class=PROV;alias=510004;mac=;phoneType=null;display=;restric=;sharedLines=
;webPass=;PrefixTrans=;melody=0;Dhcp=1;ip=;mask=;gateway=;login=;password=;
ethCX=1;nbLines=2;cTones=;timeZone=-1;langue=-
1;cWToneDisabled=0;supervisedCall=0;dustinguishMforExternal=0;forwardType=N
one;forwardNum=;forwardMsg=0;group=>

PROVEP s:2 e:510002
PROVEP T:2
#:<class=PROV;alias=510002;mac=003324420AC74;phoneType=Aastra-
6757i;display=Thom_5122;restric=;Dhcp=1;ip=;mask=;gateway=;login=5122;passw
ord=5122;ethCX=1;nbLines=2;cTones=;timeZone=-1;langue=-
1;forwardType=None;forwardNum=;forwardMsg=0;group=m2m;F1=BLF+5110+CTO;F2=BL
F+5115+PM;F3=SPEEDDIAL+5120+PDG;F4=;F5=;F6=>
<class=PROV;alias=555;mac=null;phoneType=None;display=;restric=;Dhcp=1;ip=;
mask=;gateway=;login=;password=;ethCX=1;nbLines=2;cTones=;timeZone=-
1;langue=-1;forwardType=None;forwardNum=;forwardMsg=0;group=>
<class=PROV;alias=555;mac=null;phoneType=None;display=;restric=;Dhcp=1;ip=;
mask=;gateway=;login=;password=;ethCX=1;nbLines=2;cTones=;timeZone=-
1;langue= 1;forwardType=None;forwardNum=;forwardMsg=0;group=>
```

It may be noted that in the case of the function keys, different fields are separated by plus "+".

Suppression (PROVEP s: 0)

As for the creation and consultation, PROVEP can also remove an existing profile.
PROVEP s:0 e:alias delete alias from list of profiles.

Request

PROVEP s:0 e:5100

Response

#:<**class**=PROV;**alias**=5100;**operation**=Suppression;**result**=Success

8.2. Routing

S5000 is enriched to the capabilities of managing embedded services, routes and trunks. TCP socket is used to ensure security in transactions.

8.2.1. RoutinG Embedded Service ReQuest (RGESRQ)

```
RGESRQ  T:1 s:1 NAME:NomEmbed y:Type E:maskSrce H:maskDest e: srcForward
h:destForward c:listCodecs R:Route tarn:AppliName z:sipAccount t:target
f:mediaFile
```

Creation/Destruction/Edition of Embedded Service.
Message Type: **RGESRQ**
Attribute: each attribute is optional and can be placed anywhere in the query. Unset a field is substituted by zero.

It is possible to have embedded Service without Route (Multi-domain case ...), in which case the ***R:*** flag is not set.
When route field (***R:)*** is set, it is essential to provide Route already defined in the routes table.
Index of the service in list of embedded services.

Attributes	**Definitions**
T:	TransactionID**,** present in all messages for the synchronization of Request / Response
s:	operation on embedded service : **1:Creation 2: Consultation 0: Suppression**
NAME:	name of the service
y:	Represent the type of service (FORWARD, EARLYCONNECT…)
E:	source mask
H:	destination mask
e:	source forwarded
h	destination forwarded
c	list of codec managed by endpoint using service, Different codecs are separed by coma ”,” (Ex: ***c:G711A,G729***
R:	route of the service
tarn:	name of the application connected on the service
z:	mask of sipAccount
t:	destination address
f:	media file to used depending of the type of service
p:	Index of the service in list of embedded services.

Sample

3. Création of embed sevice

RGESRQ T:4 **NAME:**TEST **s:**1 **y:** FORWARD **c:**G711A,G729 **R:**Route_OUT **H:**458* **h:**+5+* **t:**192.168.10.10 **f:**bip.sw **p:**5

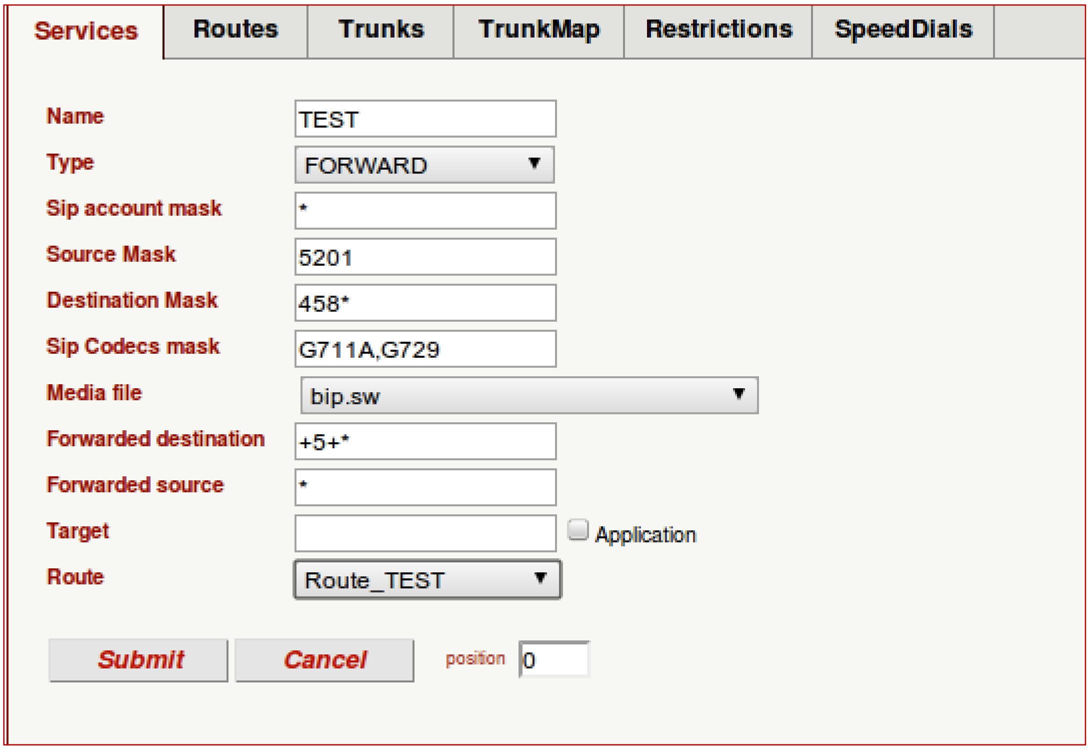
Résult: Success if ROUTE_OUT is existing route.

RGES T:1
#:<class=EmbeddedService;**name**=TEST;**operation**=Creation;**result=**Success**>**

Résult: Error if ROUTE_OUT doesn't exit yet.

RGES T:1
#:<class=EmbeddedService;**name=**TEST;**operation=**Creation;**result=**Error**>**

In the case of a successful web interface shows the table of embedded services has been enriched by a new service



4. Consultation

To ensure the creation of my service, I can make a consultation.

RGESRQ T:5 **NAME:**TEST *s:2*

The result of this query is in ASCII code and decoding gives the following representation

RGES T:2
#:<**class**=EmbeddedService;name=TEST;**sourceMask**=5201;**destMask**=458*;**destAddr**=192.168.10.10;**destFwd**=+5+*;**SrcFwd**=*;**Route**=Route_OUT;**Type**=FORWARD;**Application**=;**SipAccount**=*;**CodecsList**=G711A,G729;**Media_file**=bip.sw>

Dans le cas où le service n'existe pas une telle requête aurait comme résultat error.

RGES
T:13#:<class=EmbeddedService;name=TEST;operation=Consultation;result=**Error**>

5. Destruction

To destroy service ***(s : 0)*** just fill it's name

RGESRQ T:1 **NAME:**TEST *s:0*

8.2.2. RoutinG RouTe ReQuest (RGRTRQ)

This command creates route.

```
RGRTRQ T:tid s:operation NAME:nom_Route m:mode(0 || 1)
G:code_Error_to_continue tn: trunk_List
```

Creation/Edition/Destruction of route

MessageType: **RGRTRQ**

Attribute: each attribute is optional and can be placed anywhere in the query. Unset a field is substituted by zero.

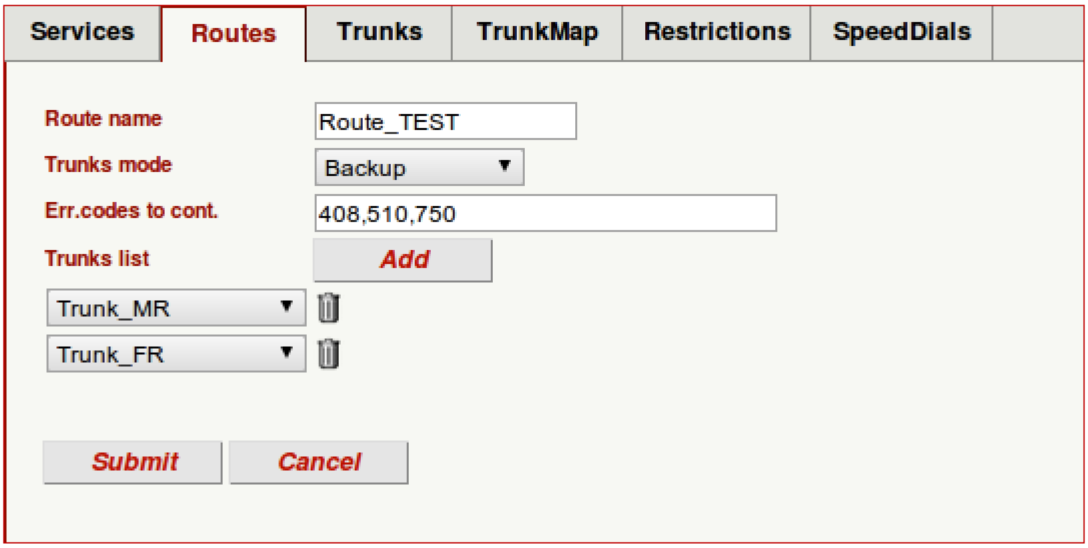
Attributes	Definitions
T:	TransactionID**,** present in all messages for the synchronization of Request / Response.
s :	**O**peration on route : **1:Creation 2: Consultation 0: Suppression**
NAME:	**R**oute name
m:	**T**taken value are **Backup** or **Loadbalanced** m: 0 => Backup m: 1 => Loadbalanced
G:	Represente list of error code. Different codes are separed by coma “,” (Ex: ***G:405,302,607,845…***)
tn:	**C**ontains all trunks, trunk list are separated by coma « , » (Ex: ***tn: trunk1, trunk2…***)

Sample:

1. **Route creation**

```
RGRTRQ T:1 NAME:Route_TEST s:1 m:0 G:408,510,750
tn:Trunk_MR,Trunk_FR
```

Result: List of routes is enriched of new route.



2. Consultation

To ensure the creation of my service, I can make a consultation. RGRTRQ **s: 2** show RGRTRQ, but in edition **(s: 2)** mode this time .

```
RGRTRQ  T:5  NAME:Route_TEST  s:2
```

The request' result is coded in Ascii which the decoding gives this following representation.

```
RGRT T:119
#:<class=Route;name=Route_TEST;mode=Backup;CodesError=408,510,750;TrunksList=Trunk_MR,Trunk_FR>
```

3. Destruction

To delete a route **(s: 0)** just fill its name.

```
RGRTRQ   T:1 NAME:Route_TEST s:0
```

If Route_TEST in any embed service result is success, else Error

```
<class=Route;name=Route_TEST;operation=Suppression;result=Success>
```

8.2.3. RoutinG TRunk ReQuest (RGTRRQ)

```
RGTRRQ T:1 s:operation NAME:Trunk_name c:listCodecs G: No_answer_Timer
H:NewDest tarn:listOfTargets op:algorithm tn: Max_of_call_accepted
```

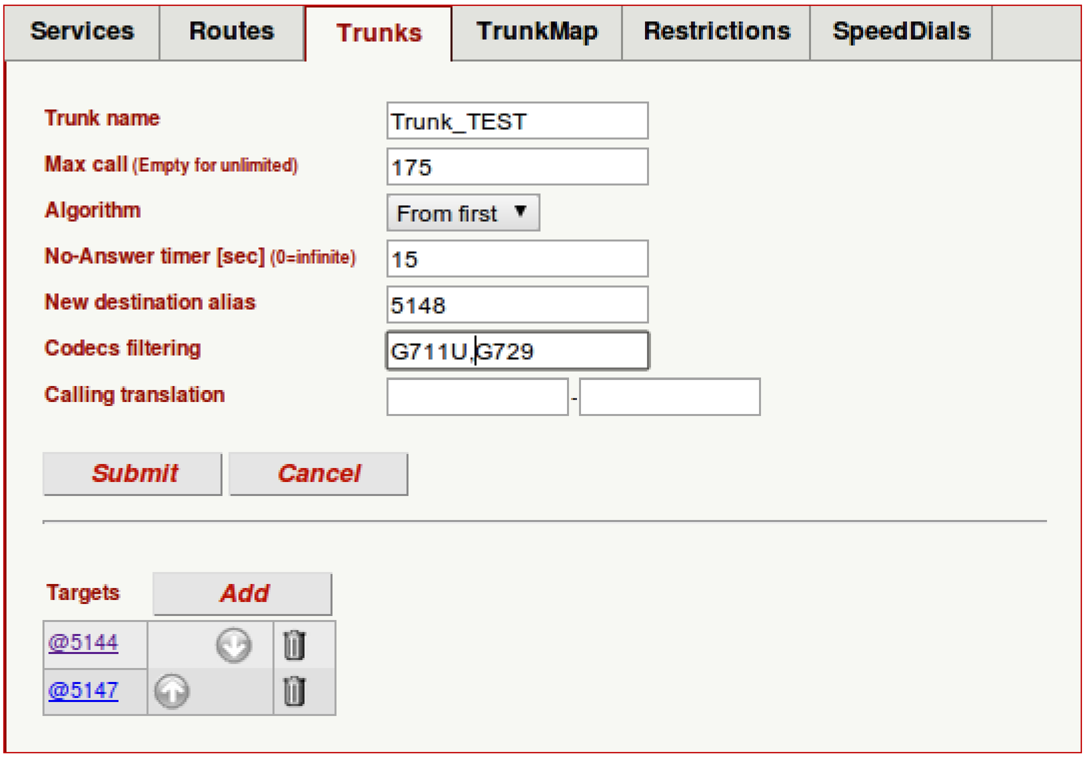
Creation/Edition/Destruction of trunk
Message Type: **RGTRRQ**
Attribute: each attribute is optional and can be placed anywhere in the query. Unset a field is substituted by zero.

attributes	definitions
T:	TransactionID**,** present in all messages for the synchronization of Request / Response
s :	Operation on route : **1:Creation 2: Consultation 0: Suppression**
NAME:	**T**runk name
c:	**L**ist of codec managed by endpoint using trunk, Different codecs are separed by coma "," (Ex: ***c:G711A,G729***)
G:	No answer timer
H	New destination field
op:	**R**epresent the algorithmic field, Taken value are: **FromFirst** or **MultiRinging**
tn:	**N**umber of call the trunk manage.
tarn:	**R**epresent list of targets of the trunk, This field is essential. If trunk contains several targets

Sample

1. Creation

```
RGTRRQ T:1 s:1 NAME:Trunk_TEST c:G711U,G729 G:15 H:5148 tarn:@5144,@5147
op:FromFirst tn:1750
RGTR T: 55 #:<class=Trunk;name=Trunk_TEST;operation=Creation;
result=Success>
```



Trunk without target

```
RGTRRQ T:1 s:1 NAME:Trunk_TEST_FAUX c:G711U,G729 G:15 H:5148  op:FromFirst
tn:1750
RGTR T:17 #:<class=Trunk;name=Trunk_TEST_FAUX;operation=Creation;
result=Error>
```

2. Consultation

```
RGTRRQ T:1 NAME:Trunk_TEST s:2
RGTR T:62
#:<class=Trunk;name=Trunk_TEST;MaxCalls=1750;Algorithme=From_first;NewDest=
5148;NoAnswerTimer=15;Targets=@5144,@5147;codecsFilters=G711U,G729>
```

3. Suppression

```
RGTRRQ T:1 NAME:Trunk_TEST s:0
RGTR T:66
#:<class=Trunk;name=Trunk_TEST;operation=Suppression;Result=Success>
```

9. Appendix

9.1. Installation of Matrix Dongles with Linux udev system

With a dongle licensing, you received a Matrix USB dongle. Before 2.6 Linux kernels, the dongle is automatically recognized after the automated installation through the “hotplug” system.
This is not the case with 2.6 and above kernels that rely on udev system.

> Contact our staff at support@m2msoft.com for up to date information and any assistance on UDEV configuration.

Some manual operations must be conducted:

6. Install the matrix library

   This is usually already done by the graphic automatic installer.
   It consists of copying libmxlin.so.2.6.0 within /usr/local/lib directory and setting two symbolic links.

```
#cp <product>/matrx/libmxlin.so.2.6.0 /usr/local/lib
#ln -s /usr/local/lib/libmxlin.so.2.6.0 /usr/local/lib/libmxlin.so.2.6
#ln -s /usr/local/lib/libmxlin.so.2.6 /usr/local/lib/libmatrix32.so
```

7. Create the « usb » group
   One must check the « usb » group exists or create it.

```
#groups
```

This displays the list of existing UNIX groups on the system

```
#m2msoft dialout cdrom audio video plugdev
```

If the group does not exist, create it as follow:

```
#groupadd usb
```

The current user must be added to the group by editing the /etc/group file.

```
#vi /etc/group
# here the group is appended at the end of the file with a user name
usb:x:1001:m2msoft
(:wq to save the file)
```

8. Udev configuration

   The access rights configuration for usb and matrix dongle is done through the /etc/udev/udev.rules file with addition of GROUP field.

```
#vi /etc/udev/udev.rules
# find the line with the word: usb_device
# and embeds in it the pattern GROUP="usb" as follow:
SUBSYSTEM=="usb_device", PROGRAM="/bin/sh -c 'K=%k; K=$${K#usbdev};
printf
bus/usb/%%03i/%%03i $${K%%%%.*} $${K#*.}'", ACTION=="add", GROUP="usb", \
NAME="%c"
```

9. Restart the host