SIP Endpoint SDK Developer's Guide

SIP Endpoint SDK 9.0.0OSX

12/19/2024

Table of Contents

Developer's Guide

This Developer's Guide contains information that will help you understand how to:

- Configure SIP Endpoint SDK for OS X
- Run the QuickStart application

Before reading this material you may want to:

- Install the SIP Endpoint SDK for OS X.
- Ensure you have access to the latest version of the API Reference.

Configuration Settings

Using the Default Configuration File

You can find the default configuration file in the following location:

```
<installation folder>/Configuration/SipEndpoint.config
```

This file contains XML configuration details that affect how your SIP Endpoint SDK application behaves. The inital settings are the same as those specified for use with the QuickStart application that is included with your SIP Endpoint SDK release.

Configuration settings are separated into two containers: the Basic Container holds the connectivity details that are required to connect to your SIP Server, while the Genesys Container holds a variety of configuration settings.

Basic Container

The first Container ("Basic") holds the basic connectivity details that are required to connect to your SIP Server. This container has at least one connection (Connectivity) element with the following attributes:

```
<Connectivity user="DN" server="SERVER:PORT" protocol="TRANSPORT"/>
```

If you are using a configuration that supports Disaster Recovery and Geo-Redundancy, there may be multiple connection elements present with each specifying a separate possible connection. Refer to the configuration settings of that feature for details. You will have to make the following changes and save the updated configuration file before using the SIP Endpoint SDK:

- `user="DN"`—Supply a valid DN for the user attribute.
- `server="SERVER:PORT"`—Replace SERVER with the host name where your SIP Server is deployed, and PORT with the SIP port of the SIP Server host. The default SIP port value is 5060. For SRV resolution, specify the SRV record without including the port number in the server's URI. Also see SRV Resolution below.
- `protocol="TRANSPORT"`—Set the protocol attribute to reflect the protocol being used to communicate with SIP Server. Possible values are UDP, TCP, or TLS.

SRV Resolution

When using an SRV record for the **server** parameter, note the following:

- SIP Endpoint SDK selects the SRV target based on the **priority** field only and does not consider the **weight** field of an SRV record.
- You can not combine IPv4 and IPv6 for a single FQDN.
- The maximum number of targets (SRV records) per service is 20.

- You can only specify SRV records in the **server** parameter of the **Connectivity** element. You can not use SRV records for the mailbox section or the **vq_report_collector** setting.

Genesys Container

The second Container ("Genesys") holds a number of configurable settings that are organized into domains and sections. These settings do not have to be changed, but can be customized to take full control over your SIP Endpoint SDK applications.

An overview of the settings in this container and the valid values for these settings is provided here:

Domain	Section	Setting
policy		
	endpoint	
		audio_qos
		include_os_version_in_user_agent_header
		include_sdk_version_in_user_agent_header
		ip_versions
		public_address
		include_mac_address
		refer_to_proxy
		rtp_inactivity_timeout
		rtp_port_binding
		rtp_port_min
		rtp_port_max
		tcp_port_min
		tcp_port_max
		signaling_qos
		sip_port_min
		sip_port_max
		sip_transaction_timeout
		video_max_bitrate
		video_qos
		vq_report_collector
		vq_report_publish
		vq_alarm_threshold
		webrtc_audio_layer

Domain	Section	Setting
		answer_sdp_priority
		sip_port_binding
		defer_device_release
	session	
		agc_mode
		auto_accept_video
		auto_answer
		auto_answer_delay
		dtmf_method
		dtmf_feedback
		echo_control
		noise_suppression
		dtx_mode
		reject_session_when_headset_na
		sip_code_when_headset_na
		vad_level
		ringback_enabled
		ringback_file
		ringing_file
		ringing_enabled
		ringing_timeout
		ringing_while_call_held
		restart_audio_if_stuck
		reject_session_when_busy
		number_sessions_for_busy

Domain	Section	Setting
		sip_code_when_busy
		rx_agc_mode
	device	
		audio_in_device For more information, see Audio Device Settings
		audio_out_device
		capture_device
		headset_name
		exclude_headset
		use_headset
		include_headset
codecs — See Working with Codec Priorities		
proxies		
	proxy''	
		display_name
		domain
		password
		reg_interval
		reg_match_received_rport
		reg_timeout
		mailbox (sub-section of proxy)
		password]]
		server

Domain	Section	Setting
		timeout
		transport
		user
		nat (sub-section of proxy)
		ice_enabled
		stun_server
		stun_server_port
		turn_password
		turn_relay_type
		turn_server
		turn_server_port
		turn_user_name
system		
	diagnostics	
		enable_logging
		log_file
		log_filter
		log_level
		log_options_provider
		log_options_endpoint
		logger_type
		log_segment
		log_expire
		log_time_convert
		log_time_format

Domain	Section	Setting
	security	
		certificate
		dylib-path
		tls_enabled
		tls-target-name-check
		use_srtp
	media	
		ringing_file

policy Domain

endpoint Section

audio_qos

Valid Values: Integer

Integer value representing the DSCP bits to set for RTP audio packets. **Note:** QoS is not supported for Windows Vista, Windows 7, or higher.

include_os_version_in_user_agent_header

Valid Values: `0`, `1`

Default Value: 1

If set to 1, the user agent field includes the OS version the client is currently running on.

include_sdk_version_in_user_agent_header

Valid Values: `0`, `1`

Default Value: 1

If set to 1, the user agent field includes the SDK version the client is currently running on.

ip_versions

Valid Values: `IPv4`, `IPv6`, `IPv4, IPv6`, `IPv6, IPv4`, or empty

Default Value: `IPv4, IPv6`

- IPv4—the application selects an available local IPv4 address; IPv6 addresses are ignored.
- IPv6—the application selects an available local IPv6 address; IPv4 addresses are ignored.

- IPv4,IPv6 or an empty—the application selects an IPv4 address if one exists. If not, an available IPv6 address is selected.
- IPv6,IPv4—the application selects an IPv6 address if one exists. If not, an available IPv4 address is selected.

Note: This parameter has no effect if the **public_address** option specifies an explicit IP address.

public_address

Valid Values: See description below

Default Value: Empty string which is fully equivalent to the `$auto` value

Local IP address or Fully Qualified Domain Name (FQDN) of the machine. This setting can be an explicit setting or a special value that the SDK uses to automatically obtain the public address.

Valid Values:

This setting may have one of the following explicit values:

- An IP address. For example, `192.168.16.123` for IPv4 or `FE80::0202:B3FF:FE1E:8329` for IPv6.
- A bare host name or fully qualified domain name (FQDN). For example, `epsipwin2` or `epsipwin2.us.example.com`.

This setting may have one of the following special values:

- $auto—The SDK selects the first valid IP address on the first network adapter that is active (status=up) and has the default gateway configured. IP family preference is specified by the policy.endpoint.ip_versions setting.
- $ipv4 or $ipv6—Same behavior as the $auto setting but the SDK restricts the address to a particular IP family.
- $host—The SDK retrieves the standard host name for the local computer using the gethostname system function.
- $fqdn—The SDK retrieves the fully qualified DNS name of the local computer. The SDK uses the GetComputerNameEx function with parameter ComputerNameDnsFullyQualified.
- An adapter name or part of an adapter name prefixed with $. For example, `$Local Area Connection 2` or `$Local`. The specified name must be different from the special values `$auto`, `$ipv4`, `$host`, and `$fqdn`.
  If the value is an explicit host name, FQDN, or $fqdn, the Contact header includes the host name or FQDN for the recipient of SIP messages (SIP Server or SIP proxy) to resolve on their own. For all other cases, including $host, the resolved IP address is used for Contact. The value in SDP is always the IP address.
- $net:subnet - The SDK will select the IP address matching the given network (from any local interface). The *subnet* is the full CIDR name as per RFC 4632. For example, `$net:192.168.0.0/16`.
- $(rule1,rule2,...) – the SDK will select the IP address from the network interface matching at least one of the given rules, where valid rules are:
  - `if='string'` – matches interface with name or description including the given string, for example:

`if='Wi-Fi'` selects the IP address from the 'Wi-Fi' adapter (if available).

- `if!='string'` – matches interface with name and description that does NOT contain the given string; intended to exclude one particular interface only, for example: `if!='Bluetooth'`.
- `net='CIDR'` – matches interface with IP address on given subnet, where CIDR is the full CIDR name as per RFC 4632, for example: net='10.0.0.0/8' selects network adapter having IP address on given local subnet.
- `net!='CIDR'` – matches interface with IP address that does NOT belong to given subnet, intended to exclude specific subnet, for example: net!='192.168.1.0/24'.
  when multiple interfaces satisfy the given rule set, local IP address selection is based on OS-defined priorities, working the same way as it does with default configuration.

include_mac_address

Valid Values: `0`, `1`

Default Value: `0`

If set to 1, the MAC address is included in the Contact header of the REGISTER message of the host's network interface in a format compatible with RFC 5626.

refer_to_proxy

Valid Values: `0`, `1`

Default Value: `0`

Specifies the destination of a referred INVITE.

- `0`—Send the INVITE to the URL specified in the Refer-To header of the REFER message.
- `1`—Send the INVITE to your configured SIP Proxy.

rtp_inactivity_timeout

Valid Values: `5-150`

Default Value: `150`

Suggested Value: `30`

Timeout interval in seconds for RTP inactivity.

rtp_port_binding

Valid Values: `0,1`

Specifies how SIP Endpoint binds the RTP port:

- 0 — opens the RTP and RTCP ports to listen on any network interface.
- 1 — the RTP and RTCP ports bind to the interface specified by the **public_address** setting and listen only on that IP address.

Important: When **rtp_port_binding** is set to `0`, the **ip_versions** option must specify a single IP version, otherwise a wrong protocol version might be selected for outgoing UDP packages. Default value: 1

rtp_port_min

Valid Values: `9000-65535`

The integer value representing the minimum value for an RTP port range. Must be within the valid port range of `9000` to `65535`. If the minimum and maximum values are not specified or are set to an invalid value, the default minimum (9000) and maximum (minimum value + 999) are used. Setting the minimum to a value that is larger than the maximum is considered an error and will result in a failure to initialize the endpoint.

rtp_port_max

Valid Values: `9000-65535`

The integer value representing the maximum value for an RTP port range. Must be within the valid port range of `9000` to `65535`. If the minimum and maximum values are not specified or are set to an invalid value, the default minimum (9000) and maximum (minimum value + 999) are used. Setting the maximum to a value that is less than the minimum is considered an error and will result in a failure to initialize the endpoint.

tcp_port_min

Valid Values: `0-65535`

The integer value representing the minimum value for a TCP client-side port range. Must be within the valid port range of `1` to `65535`. If set to `0` (default) or if the configured range is not valid, SIP connections over TCP and TLS use ephemeral ports, assigned by the operating system.

tcp_port_max

Valid Values: `0-65535`

The integer value representing the maximum value for a TCP client-side port range. Must be within the valid port range of `1` to `65535`. If set to `0` (default) or if the configured range is not valid, SIP connections over TCP and TLS use ephemeral ports, assigned by the operating system.

If the value is non-zero and greater than the *tcp_port_min* value, this value specifies the maximum value for a TCP client-side SIP port range that will be used for all outgoing SIP connections over TCP and TLS transport.

signaling_qos

Valid Values: Integer

The integer value representing the DSCP bits to set for SIP packets. **Note:** QoS is not supported for Windows Vista, Windows 7, or higher.

sip_port_min

Valid Values: `1-65535`

The integer value representing the minimum value for a SIP port range. Must be within the valid port range of 1 to 65535. If the minimum and maximum values are not specified or are set to an invalid value, the default minimum (5060) and maximum (minimum value + 6) are used. Setting the minimum to a value that is larger than the maximum is considered an error and will result in a failure to initialize the endpoint.

sip_port_max

Valid Values: `1-65535`

The integer value representing the maximum value for a SIP port range. Must be within the valid port range of 1 to 65535. If the minimum and maximum values are not specified or are set to an invalid value, the default minimum (5060) and maximum (minimum value + 6) are used. Setting the maximum to a value that is less than the minimum is considered an error and will result in a failure to initialize the endpoint.

sip_transaction_timeout

Valid Values: `1-32000`

Default Value: `4000`

SIP transaction timeout value in milliseconds. Valid values are 1 through 32000, with a default value of 4000. The recommended value is 4000.

video_max_bitrate

Valid Values: Integer

Integer value representing the maximum video bitrate.

video_qos

Valid Values: Integer

The integer value representing the DSCP bits to set for RTP Video packets. **Note:** QoS is not supported for Windows Vista, Windows 7, or higher.

vq_report_collector

See Producing RTCP Extended Reports.

vq_report_publish

See Producing RTCP Extended Reports.

vq_alarm_threshold

Valid Values: `0` or a number from `1.0` to `5.0`

Default Value: `0`

Specifies the MOS threshold for generating Voice Quality Alarms. A `0` value disables the alarms. The recommended threshold value is `3.5`. Genesys recommends that you avoid using values above `4.2` as an MOS that high might not be obtainable with some codecs, even in perfect network conditions.

webrtc_audio_layer

Valid Values: `0`, `1`, `2`, `1000`, `2000`, `3000`

Default Value: `0`

Specifies which audio layer is used for WebRTC.

- `0` — The audio layer is defined by the `GCTI_AUDIO_LAYER` environment variable — Core audio is used if this environment variable is not specified.
- `1` — Wave audio layer is used.

- 2 — Core audio layer is used.
- `1000` — Instructs the audio layer to open the microphone channel when the endpoint starts up, using the audio layer type defined by option `0`, and to keep it open until the endpoint is terminated.
- `2000` — Opens the speaker channel for the life of the endpoint, using the audio layer type defined by option `0`. Eliminates any delay in opening the audio device when an incoming or outgoing call is connected, for example in environments where audio device startup is slow due to a required restart of the Windows MMCSS service.
- `3000` — Opens the microphone and speaker channels for the life of the endpoint, using the audio layer type defined by option `0`.

Important

Keeping the audio channels permanently open eliminates any delay in connecting audio device to the call works around any issues with device occasionally not starting (or stopping) properly, at the cost of very small performance penalty.

answer_sdp_priority

Valid Values: `config`, `offer`

Default Value: `config`

- `config`—the endpoint selects the first codec from the codec configuration listed in both the codec configuration and the SDP offer.
- `offer`—the endpoint selects the first codec in the SDP offer listed in both the codec configuration and the SDP offer.

sip_port_binding

Valid Values: `0`, `1`

Default Value: `0`

- `0`—opens the SIP port to listen on any interface.
- `1`—the SIP port binds to the interface specified by the `public_address` setting and listens only on this IP address.

Important

The **sip_port_binding** must be set to `0` for automatic IP address change detection to work properly. When set to 1 for specific deployments, agents have to re-register connections manually after an IP address change.

defer_device_release

Valid Values: Any integer

Default Value: `200`

If set to a non-zero value, releasing of audio devices will be deferred for a given time (in milliseconds) after the audio stream has been stopped, to avoid any potential service interruptions when the audio is going to be quickly restarted, and if audio device operations are too slow on the user workstation or have other problems with restart. A zero value disables the deferred device release.

session Section

agc_mode

Valid Values: `0`, `1`

Default Value: `1`

If set to 0, AGC (Automatic Gain Control) is disabled; if set to 1, it is enabled. Other values are reserved for future extensions. This configuration is applied at startup, after which time the **agc_mode** setting can be changed to 1 or 0 from the main sample application.

Note: It is not possible to apply different AGC settings for different channels in multi-channel scenarios.

auto_accept_video

Valid Values: `0`, `1`

This setting is only used in auto-answer scenarios when `auto_answer=1`.

If auto_accept_video is set to 1, both audio and video streams are accepted, otherwise incoming calls are answered as audio only, even if video is present in the offer.

auto_accept_video applies to a 3pcc answer when make-callrfc3275 is configured to 1 on the originating DN and a video codec is configured in the endpoint. auto_accept_video is not applied to a 3pcc answer when make-call-rfc3275 is configured to 2 on an originating DN, even if auto_accept_video is set to 1 and a video codec is configured in the endpoint.

auto_answer

Valid Values: `0`, `1`

If set to 1, all incoming calls should be answered automatically.

auto_answer_delay

Valid Values: Number in milliseconds

Default Value: `0`

Time in milliseconds to wait before auto-answering (only applicable when `auto_answer=1` and `auto-answer` is not blocked by missing headset).

callwait_tone_enabled

Valid Values: 0, 1

Default Value: 0

Specifies whether the call waiting tone is enabled for incoming calls, to be played when a new call arrives while user is on active call.

callwait_tone_file

Valid Values: Empty or the path to the call waiting tone sound file. The path can be a file in the current directory or the full path to the sound file.

Default Value: Empty

Specifies the audio file that is played when the call waiting tone is enabled with the `callwait_tone_enabled` option.

The call waiting tone file must be a RIFF (little-endian) WAVE file using one of the following formats:

- kWavFormatPcm = 1, PCM, each sample of size bytes_per_sample
- kWavFormatALaw = 6, 8-bit ITU-T G.711 A-law (8 KHz sampling rate)
- kWavFormatMuLaw = 7, 8-bit ITU-T G.711 mu-law (8 KHz sampling rate)

Uncompressed PCM audio must be 16-bit mono or stereo with a sampling rate of 8, 16, or 32 KHz.

dtmf_feedback

Valid Values: 0 (disable feedback), 1 (enable feedback)

Default Value: 1

Controls whether local feedback tone is played when Endpoint sends DTMF to remote party.

dtmf_method

Valid Values: `Rfc2833`, `Info`, `InbandRtp`

Method to send DTMF.

echo_control

Valid Values: `0`, `1`

If set to 1, echo control is enabled.

noise_suppression

Valid Values: `0`, `1`

If set to 1, noise suppresion is enabled.

dtx_mode

Valid Values: `0`, `1`

If set to 1, DTX is activated.

reject_session_when_headset_na

Valid Values: `0`, `1`

If the SDK is configured to use the headset setting (`policy.device.use_headset`=1) and the `reject_session_when_headset_na` option is set to 1, the SDK should reject the incoming session if a

USB headset is not available.

sip_code_when_headset_na

Valid Values: SIP Error Code

Default Value: `480`

If a valid SIP error code is supplied, the SDK rejects the incoming session with the specified SIP error code if a USB headset is not available.

vad_level

Valid Values: `0`-3

Sets the degree of bandwidth reduction, from `0` for conventional VAD to 3 for aggressive high.

ringback_enabled

Valid Values: `0`, 1, 2, 3, 4, 6

Default Value: 2

Specifies whether the ringback tone is enabled for outgoing calls.

- `0` — The ringback is not played when the INVITE dialog is not yet established. In scenarios where ringback is provided by Media Server, the ringback tone would be still present.
- 1 — The incoming media stream is played if provided by the Media gateway in a reliable provisional response with SDP.
- 2 — A local file is used for the ringback.
- 3 — The ringback is always played using either a local file or media provided by the gateway, if the provisional response is reliable.
- 4 — Same as 1, but the incoming media stream is played even if the provisional response from Media gateway is not reliable.
- 6 — The ringback is always played using either a local file or media provided by the gateway (regardless of whether the provisional response is reliable or not).

ringback_file

Valid Values: Empty or the path to the ringback sound file. The path can be a file in the current directory or the full path to the sound file.

Default Value: Empty

Specifies the audio file that is played when the ringing tone is enabled with the `ringing_enabled` option.

WebRTC does not support MP3 playback. The ringtone file for built-in ringback must be a RIFF (little-endian) WAVE file using one of the following formats:

- kWavFormatPcm = 1, PCM, each sample of size bytes_per_sample
- kWavFormatALaw = 6, 8-bit ITU-T G.711 A-law (8 KHz sampling rate)
- kWavFormatMuLaw = 7, 8-bit ITU-T G.711 mu-law (8 KHz sampling rate)

Uncompressed PCM audio must be 16-bit mono or stereo with a sampling rate of 8, 16, or 32 KHz.

ringing_enabled

Valid values: An integer between 0 to 7.
Default Value: 3

- 0—event Ringing disabled
- 1—event Ringing enabled
- 2—play ringtone internally (event Ringing disabled)
- 3—play ringtone internally and enable event Ringing.
- 4—play ringtone through a separate ringer device.
- 5—play ringtone through the ringer device and enable event Ringing.
- 6—play ringtone internally once for full duration (**ringing_timeout** is not used and the ringing does not stop when call is answered).
- 7—play ringtone once for full duration and enable event Ringing.

Important

If the ringtone is configured to play for full duration (values *6* or *7*), then it will be played even in case of auto-answer with zero delay (configured either in SIP Endpoint SDK or desktop application), unless it is suppressed. This configuration setting assumes **ringing_file** option being configured with a short "beep" sound that would not interfere much with agent hearing the customer (as the ringtone will be played in parallel / mixed with remote audio stream).

Suppressing the Ringtone

The ringtone is generated for all incoming calls to the Genesys SIP Endpoint SDK. To suppress the ringtone for third-party call control for the originating DN, configure the following SIP Server option:

- `make-call-alert-info=<urn:alert:service:3pcc@genesys>`

or

- `make-call-alert-info=<file://null>;service=3pcc`

> **Important**
>
> If at least one application based on SIP Endpoint in the contact center is configured with the `ringing_enabled` option set to a non-zero value, the SIP Server `make-call-alert-info` option should be set to one of the specified values.

ringing_timeout

Valid Values: Empty, `0`, or a positive number

Default Value: `0`

Specifies the duration, in seconds, of the ringing tone. If set to `0` or if the value is empty, the ringing time is unlimited.

ringing_file

Valid Values: Empty or the path to the ringing sound file. The path may be a file name in the current directory or the full path to the sound file.

Default Value: ringing.wav

Specifies the audio file that is played when the ringing tone is enabled with the `ringing_enabled` option.

Note that WebRTC does not support MP3 playback. The ringtone file for built-in ringing should be a RIFF (little-endian) WAVE file using one of the following formats:

- kWavFormatPcm = 1, PCM, each sample of size bytes_per_sample
- kWavFormatALaw = 6, 8-bit ITU-T G.711 A-law
- kWavFormatMuLaw = 7, 8-bit ITU-T G.711 mu-law

Uncompressed PCM audio must 16 bit mono or stereo and have a frequency of 8, 16, or 32 KHZ.

ringing_while_call_held

Valid Values: `0` or 1

Default Value: 1
If set to `0`, when another call is held, playing ringtone is suppressed and call wait tone is played instead (if configured). If set to 1 (default value), ringtone is played whenever a new call arrives and there are no other active calls; held calls are not considered active in this case.

restart_audio_if_stuck

Valid Values: Empty, 0, 1

Default Value: 0

- 0 or Empty—disable auto restart for stuck audio
- 1—enable auto restart for stuck audio

reject_session_when_busy

Valid Values: Empty, 0, 1

Default Value: 0

- 0 or Empty—disable rejection of a session when busy
- 1—enable rejection of a session when busy

number_sessions_for_busy

Valid Values: Positive integer

Default Value: 1

Sets the number of sessions before busy. Must be a positive integer.

sip_code_when_busy

Valid Values: Empty, 4xx, 5xx, 6xx

Default value: Empty

SIP error response code to use when busy. Can be set to any valid SIP error response code in the 4xx, 5xx, or 6xx range, for example, 486.

rx_agc_mode

Valid Values: 0, 1

Default value: 0

When set to 1, the SDK enables the receiving-side AGC allowing the volume of the received RTP stream to be adjusted automatically. When set to `0` (default), the feature is disabled.

device Section

audio_in_device

Valid Values: A regex that matches the ECMAScript standard.

Microphone device name.

For more information, see Audio Device Settings

audio_out_device

Valid Values: A regex that matches the ECMAScript standard.

Speaker device name.

capture_device

Valid Values: A regex that matches the ECMAScript standard.

Capture device name.

headset_name

Valid Values: A regex that matches the ECMAScript standard.

The name of the headset model.

exclude_headset

Valid Values: A regex that matches the ECMAScript standard.

Default Value: Empty

When set, this will exclude devices whose name matches the regex pattern from being considered a valid headset for automatic device selection. Microphone and speaker parts of the excluded headset can still be selected manually (or even automatically, if no better device is found), but such a selection will not set "headset available" flag.

use_headset

Valid Values: `0`, `1`

If set to `0`, the audio devices specified in audio_in_device and audio_out_device are used by the SDK.

If set to 1, the SDK uses a headset as the preferred audio input and output device and the audio devices specified in audio_in_device and audio_out_device are ignored.

include_headset

Valid Values: Pair of device names or name parts with microphone and speaker names separated by colon, or comma-separated list of such pairs, for example: `External Mic : Headphones`

Default Value: Empty

When set, specifies the list of audio in/out devices to be considered as headset for automatic device selection, applicable to case when *use_headset = 1*. The names including delimiter character (quotes, colon or comma) must be enclosed in single or double quotes.

codecs Domain

See Working with Codec Priorities

proxies Domain

Configure a proxy section for each connectivity line. For example, for three connectivity lines, configure sections for `proxy0`, `proxy1`, and `proxy2`.

When the proxy section does not exist in the configuration file for a particular connectivity line, the framework takes the configurations settings from the `proxy0` section. You can use this feature in use cases where the proxy sections are the same for all connectivity lines.

proxy Section

display_name

Valid Values: String

Proxy display name.

domain

Valid Values:Any valid SIP domain

Default Value: Empty

A SIP domain is an application layer configuration defining the management domain of a SIP proxy. The configured value should include hostport and may include uri-parameters as defined by RFC 3261. The scheme, userinfo, and transport URI parameters are included automatically.

If set to an empty string, SIP Endpoint SDK for .NET uses the parameters from the Connectivity section to construct the SIP domain value as it did in previous versions.

password

Valid Values: String

Proxy password.

reg_interval

Valid Values: Integer

Default Value: `0`

The period, in seconds, after which the endpoint starts a new registration cycle when a SIP proxy is down. Valid values are integers greater than or equal to `0`. If the setting is empty or negative, the default value is 0, which means no new registration cycle is allowed. If the setting is greater than 0, a new registration cycle is allowed and will start after the period specified.

reg_match_received_rport

Valid Values: `0` or 1

Default Value: `0`

This setting controls whether or not SIP Endpoint SDK should re-register itself when receiving an IP address (in the `received` parameter of a REGISTER response) that is different from the address supplied in the Contact header and does not match any local network interfaces. A value of `0` (default) disables this feature and a value of 1 enables re-registration.

Starting from 9.0.003, this setting is deprecated and is not recommended for use, unless suggested by Genesys Technical Support to fix specific problems. When the `received` parameter of a REGISTER response matches a local IP address, changing the IP address and re-registering is now done automatically.

reg_timeout

Valid Values: Number in seconds

The period, in seconds, after which registration should expire. A new REGISTER request will be sent before expiration. Valid values are integers greater than or equal to 0. If the setting is `0` or empty/null, then registration is disabled, putting the endpoint in standalone mode.

mailbox Sub-section

Important

mailbox is a sub-section of the **proxy<n>** section.

password

Valid Values: String

Mailbox password.

server

Valid Values: String

Proxy server address and port for this mailbox.

timeout

Valid Values: Number in seconds

Default Value: `1800`

Subscription expiration timeout in seconds. If the setting is missing or set to `0`, the SDK uses a default timeout of `1800` seconds (30 minutes).

transport

Valid Values: `udp`, `tcp`, `tls`

Transport protocol to use when communicating with the server.

user

Valid Values: String

Mailbox ID for this mailbox.

nat Sub-section

> **Important**
> **nat** is a sub-section of the **proxy<n>** section.

ice_enabled

Valid Values: Boolean

Enable or disable ICE.

stun_server

Valid Values: String

STUN server address. An empty or null value indicates this feature is not used.

stun_server_port

Valid Values: Valid port number

Default Value: 3478

STUN server port value.

turn_password

Valid Values: String

Password for TURN authentication.

> **Warning**
> Starting from 9.0.012.02, this setting is deprecated and is not recommended for use, unless suggested by Genesys Technical Support to fix specific problems. Use the GCTI_TURN_PASSWORD environment variable to set the password for TURN authentication.

turn_relay_type

Valid Values: `0`, udp, 1, or `tcp`

Type of TURN relay.

- `0` or udp for TURN over UDP.
- 1 or `tcp` for TURN over TCP.

turn_server

Valid Values: String

TURN server address. An empty or null value indicates this feature is not used.

turn_server_port

Valid Values:Valid port number

Default Value: 3478

TURN server port value.

turn_user_name

Valid Values: String

User ID for TURN authorization

Warning
Starting from 9.0.012.02, this setting is deprecated and is not recommended for use, unless suggested by Genesys Technical Support to fix specific problems. Use the GCTI_TURN_USERNAME environment variable to set the username for TURN authentication.

`system` Domain

diagnostics Section

enable_logging

Valid Values: `0` or 1

Default Value: 1

Disable or enable logging.

log_file

Valid Values: String

Log file name, for example, `SipEndpoint.log`.

log_filter

Valid Values: *Empty*, `dtmf`

Default Value: *Empty*

Specifies the list of log filters to be applied to hide sensitive data from the endpoint log. Currently the only supported filter is `dtmf`, which hides all occurrences of DTMF data from the log (by replacing entered digits with 'x').

log_level

Valid Values: `0`-4

Default Value: 3

Log levels: `0` = "Fatal"; 1 = "Error"; 2 = "Warning"; 3 = "Info"; 4 = "Debug"

log_options_provider

Valid Values: Comma-separated list of log setting for various low-level components, including:

- `gsip=N` - Genesys SIP library log level, default is 2
- `webrtc=(level)` - Log level(s) for third-party WebRTC native code component. The default is `error` to include error messages and important WebRTC diagnostic, value `api` adds low-level API printouts.

Example value: `gsip=2,webrtc=(error)`

Warning

This settings control low-level debug information used for troubleshooting. Please don't change the value unless instructed by Genesys Technical Support engineer.

log_options_endpoint

Valid Values: `0`-4, same as **log_level**

Default Value: 2

Log levels: `0` = "Fatal"; 1 = "Error"; 2 = "Warning"; 3 = "Info"; 4 = "Debug"

5 = Logging disabled.

This setting should not be set higher than log_level setting.

logger_type

Valid Values: file

If set to `file` the log data will be printed to the file specified by the **log_file** value.

log_segment

Valid Values: `false`, number, or number in KB,MB, or hr

Default Value: `10 MB`

- false: No segmentation is allowed
- or KB: Size in kilobytes
- MB: Size in megabytes
- hr: Number of hours for segment to stay open

Specifies the segmentation limit for a log file. If the current log segment exceeds the size set by this option, the file is closed and a new one is created. This option is ignored if log output is not configured to be sent to a logfile.

log_expire

Valid Values: `false`, number, number `file`, number day

Deafult Value: `10` (store 10 log fragments and purge the rest)

- false: No expiration; all generated segments are stored.
- or file: Sets the maximum number of log files to store. Specify a number from 1—1000.
- day: Sets the maximum number of days before log files are deleted. Specify a number from 1—100

Determines whether log files expire. If they do, sets the measurement for determining when they expire, along with the maximum number of files (segments) or days before the files are removed. This option is ignored if log output is not configured to be sent to a log file.

log_time_convert

Valid Values: `local`, `utc`

Default Value: `local`

- `local`: The time of log record generation is expressed as a local time, based on the time zone and any seasonal adjustments. Time zone information of the application’s host computer is used.
- `utc`: The time of log record generation is expressed as Coordinated Universal Time (UTC).

Specifies the system in which an application calculates the log record time when generating a log file. The time is converted from the time in seconds since the Epoch (00:00:00 UTC, January 1, 1970).

log_time_format

Valid Values: `time`, `locale`, `ISO8601`

Default Value: time

- `time`: The time string is formatted according to the HH:MM:SS.sss (hours, minutes, seconds, and milliseconds) format
- `locale`: The time string is formatted according to the system’s locale.
- `ISO8601`: The date in the time string is formatted according to the ISO 8601 format. Fractional seconds are given in milliseconds.

Specifies how to represent, in a log file, the time when an application generates log records. A log record’s time field in the ISO 8601 format looks like this: 2001-07-24T04:58:10.123.

security Section

Important

SIP Endpoint SDK no longer uses the **tls_enabled** setting.

certificate

See Configuring certificate option for TLS.

dylib-path

Valid Values: Empty, path to dynamic library location
Default Value: Empty

Specifies the location of Genesys security module's dynamic libraries for TLS support. The name of the dynamic library in different processors are:

- `libgsecurity_openssl_64.dylib` for Intel processor
- `libgsecurity_openssl_arm64.dylib` for Apple processor

If the option is not specified or set to empty value (default), SIP Endpoint SDK requires the dynamic libraries to be located in the **Resources** folder of the application's main bundle

tls-target-name-check

See Configuring tls-target-name-check option for TLS.

use_srtp

Valid Values: `optional`, `allowed`, `disabled`, `off`, `elective`, `both`, `enabled`, `force`, `mandatory`

Indicates whether to use SRTP:

- `optional` or `allowed`—do not send secure offers, but accept them
- `disabled` or `off`—do not send secure offers and reject incoming secure offers
- `elective` or `both`—send both secure and non-secure offers and accept either
- `enabled`—send secure offers, accept both secure and non-secure offers
- `force` or `mandatory`—send secure offers, reject incoming non-secure offers

Adding either `',UNENCRYPTED_SRTCP'` (long form) or `',UEC'` (short form) to any value (for example, "*enabled,UEC*`""`), would result in the **UNENCRYPTED_SRTCP** parameter being added to that offer. When this parameter is negotiated, RTCP packets are not encrypted, but are still authenticated.

media Section

ringing_file

Valid Values: Empty, String file name

Default Value: `ringing.mp3`

The Ringing sound file name in the current directory or the full local path to the ringing sound file.

Additional Configuration Options

The default configuration file may not contain all settings that may be used with the SIP Endpoint SDK; additional settings can be added to change certain behaviors. Check Configuring SIP Endpoint SDK for .NET for a discussion of these additional settings.

Configuring SIP Endpoint SDK for OS X

Using the Default Configuration File

You can find the default configuration file in the following location:

```
<installation folder>/Bin/SipEndpoint.config
```

This file contains XML configuration details that affect how your SIP Endpoint SDK application behaves. The initial settings are the same as those specified for use with the QuickStart application included with your SIP Endpoint SDK release.

Configuration settings are separated into two containers: the Basic Container holds the connectivity details that are required to connect to your SIP Server, while the Genesys Container holds a variety of configuration settings.

Basic Container

The first Container ("Basic") holds the basic connectivity details that are required to connect to your SIP Server. This container has at least one connection (Connectivity) element with the following attributes:

```
<Connectivity user="DN" server="SERVER:PORT" protocol="TRANSPORT"/>
```

Make the following changes and save the updated configuration file before using the SIP Endpoint SDK:

- **user="DN"** — Supply a valid DN for the user attribute.
- **server="SERVER:PORT"** — Replace SERVER with the host name where your SIP Server is deployed, and PORT with the SIP port of the SIP Server host. (The default SIP port value is 5060.) For SRV resolution, specify the SRV record without including the port number in the server's URI.
- **protocol="TRANSPORT"** — Set the protocol attribute to reflect the protocol being used to communicate with SIP Server. Possible values are UDP, TCP, or TLS.

SRV Resolution

When using an SRV record for the **server** parameter, note the following:

- SIP Endpoint SDK selects the SRV target based on the **priority** field only and does not consider the **weight** field of an SRV record.
- You can not combine IPv4 and IPv6 for a single FQDN.
- The maximum number of targets (SRV records) per service is 20.
- You can only specify SRV records in the **server** parameter of the **Connectivity** element. You can not

use SRV records for the mailbox section or the **vq_report_collector** setting.

Genesys Container

The second Container ("Genesys") holds a number of configurable settings that are organized into domains and sections. These settings do not have to be changed, but can be customized to take full control over your SIP Endpoint SDK applications.

An overview of the settings in this container and the valid values for these settings is provided here:

Domain	Section	Setting
policy		
	endpoint	
		audio_qos
		include_os_version_in_user_agent_header
		include_sdk_version_in_user_agent_header
		ip_versions
		public_address
		rtp_inactivity_timeout
		rtp_port_min
		rtp_port_max
		signaling_qos
		sip_port_min
		sip_port_max
		sip_transaction_timeout
		vq_report_collector
		vq_report_publish
		vq_alarm_threshold
		answer_sdp_priority
		sip_port_binding
		video_max_bitrate
		video_qos
		defer_device_release
	session	
		agc_mode
		auto_accept_video
		auto_answer
		auto_answer_delay
		dtmf_method

Domain	Section	Setting
		dtmf_feedback
		echo_control
		noise_suppression
		dtx_mode
		reject_session_when_headset_na
		sip_code_when_headset_na
		vad_level
		ringback_enabled
		ringback_file
		ringing_enabled
		ringing_timeout
		ringing_file
		ringing_while_call_held
		reject_session_when_busy
		number_sessions_for_busy
		sip_code_when_busy
rx_agc_mode		
	device	
		audio_in_device For more information, see Audio Device Settings
		audio_out_device
		capture_device
		ringer_device
		headset_name
		exclude_headset
		use_headset
		include_headset
codecs—See Working with Codec Priorities		
proxies		
	proxy	
		display_name
		domain
		password
		reg_interval
		reg_match_received_rport
		reg_timeout

Domain	Section	Setting
		mailbox
		server
		timeout
		transport
		user
		nat (sub-section of proxy)
		ice_enabled
		stun_server
		stun_server_port
		turn_password
		turn_relay_type
		turn_server
		turn_server_port
		turn_user_name
system		
	diagnostics	
		enable_logging
		log_file
		log_filter
		log_level
		log_options_provider
		logger_type
	security	
		certificate
		certificate-key
		trusted-ca
		sec-protocol
		cipher-list
		ciphersuites
		tls-crl
		tls-target-name-check
		use_srtp

`policy` Domain

`endpoint` Section

audio_qos

Valid Values: Number

The integer value representing the DSCP bits to set for RTP audio packets. The value should be `4 * Preferred DSCP value`.

include_os_version_in_user_agent_header

Valid Values: `0` or `1`
Default Value: `1`

If set to 1, the user agent field includes the OS version the client is currently running on.

include_sdk_version_in_user_agent_header

Valid Values: `0` or `1`
Default Value: `1`

If set to 1, the user agent field includes the SDK version the client is currently running on.

ip_versions

Valid Values: IPv4, empty Default Value: `IPv4`.

A value of `IPv4` or an empty value means that the application selects an available local IPv4 address.

NOTE: This parameter has no effect if the `public_address` option specifies an explicit IP address.

public_address

Valid Values: See description below.

Local IP address of the machine. This setting can be an explicit setting or a special value that the SDK uses to automatically obtain the public address.

Valid Values:

This setting may have one of the following explicit values:

- An IP address. For example, `192.168.16.123` for IPv4.
- A bare host name, for example, `epsipmac2`.

Default Value: Empty string which is fully equivalent to the `$auto` value.
This setting may have one of the following special values:

- `$auto`—The SDK selects the first valid IP address on the first network adapter that is active (status=up) and has the default gateway configured. IP family preference is specified by the **policy.endpoint.ip_versions** setting.
- `$ipv4`—Same behavior as the `$auto` setting but the SDK restricts the address to a particular IP family.
- `$host`—The SDK retrieves the standard host name for the local computer using the `gethostname` system function.
- An adapter name or part of an adapter name prefixed with `$`. For example, `en0` or `en1`. The specified name must be different from the special values `$auto`, `$ipv4`, and `$host`.
- $net:subnet - The SDK will select the IP address matching the given network (from any local interface). The *subnet* is the full CIDR name as per RFC 4632. For example, `$net:192.168.0.0/16`.

If the value is specified as an explicit host name, the `Contact` header includes the host name for the recipient of SIP messages (SIP Server or SIP proxy) to resolve on their own. For all other cases, including `$host`, the resolved IP address is used for `Contact`. The value in SDP is always the IP address.

rtp_inactivity_timeout

Valid Values: Integer between 5 and `150`
Default Value: `150`
Suggested Value: `30`

Timeout interval in seconds for RTP inactivity.

rtp_port_min

Valid Values: Number
Default Value: `8000`

The integer value representing the minimum value for an RTP port range. Must be within the valid port range of `8000` to `65535`. If the minimum value is not specified or set to an invalid value, the default value of `8000` is used for `rtp_port_min`. Setting the minimum to a value that is larger than the maximum is considered an error and will result in a failure to initialize the endpoint.

rtp_port_max

Valid Values: Number
Default Value: `9000`

The integer value representing the maximum value for an RTP port range. Must be within the valid port range of `8000` to `65535`. If the maximum value is not specified or set to an invalid value, the default maximum `9000` is used for `rtp_port_max`. Setting the maximum to a value that is less than the minimum is considered an error and results in a failure to initialize the endpoint.

signaling_qos

Valid Values: Number

The integer value representing the DSCP bits to set for SIP packets. Integer value should be configured to `4 * Preferred DSCP value`.

sip_port_min

Valid Values: Number between 1 and `65535`
Default Value: `5060`

The integer value representing the minimum value for a SIP port range. Must be within the valid port range of `1` to `65535`. If the minimum value is not specified or set to an invalid value, the default value of `5060` is used for `sip_port_min`. Setting the minimum to a value that is larger than the maximum is considered an error and will result in a failure to initialize the endpoint.

sip_port_max

Valid Values: Number between 1 and `65535`
Default Value: `5080`

The integer value representing the maximum value for a SIP port range. Must be within the valid port range of `1` to `65535`. If the maximum value is not specified or set to an invalid value, the default value of `5080` is used for `sip_port_max`. Setting the maximum to a value that is less than the minimum is considered an error and will result in a failure to initialize the endpoint.

sip_transaction_timeout

Valid Values: Number between 1 and `32000`
Default Value: `4000`

SIP transaction timeout value in milliseconds. Valid values are `1` through `32000`, with a default value of `4000`. The recommended value is `4000`.

vq_report_collector

See Producing RTCP Extended Reports

vq_report_publish

See Producing RTCP Extended Reports

vq_alarm_threshold

Valid Values: `0` or a number from `1.0` to `5.0`

Default Value: `0`

Specifies the MOS threshold for generating Voice Quality Alarms. A `0` value disables the alarms. The recommended threshold value is `3.5`. Genesys recommends that you avoid using values above `4.2` as an MOS that high might not be obtainable with some codecs, even in perfect network conditions.

answer_sdp_priority

Valid Values: `config` or `offer`
Default value: `config`

- `config`—The endpoint selects the first codec from the codec configuration listed in both the codec configuration and the SDP offer.
- `offer`—the endpoint selects the first codec in the SDP offer listed in both the codec configuration and the SDP offer.

sip_port_binding

Valid Values: `0` or `1`
Default Value: `0`

- `0`—opens the SIP port to listen on any interface
- `1`—the SIP port binds to the interface specified by the `public_address` setting and listens only on this IP address.

Important

The **sip_port_binding** must be set to `0` for automatic IP address change detection to work properly. When set to 1 for specific deployments, agents have to re-register connections manually after an IP address change.

video_max_bitrate

Valid Values: Integer

Integer value representing the maximum video bitrate.

video_qos

Valid Values: Integer

The integer value representing the DSCP bits to set for RTP Video packets.

defer_device_release

Valid Values: Any integer

Default Value: `200`

If set to a non-zero value, releasing of audio devices will be deferred for a given time (in milliseconds) after the audio stream has been stopped, to avoid any potential service interruptions when the audio is going to be quickly restarted, and if audio device operations are too slow on the user workstation or have other problems with restart. A zero value disables the deferred device release.

`session` Section

agc_mode

Valid Values: `0` or `1`

If set to `0`, AGC (Automatic Gain Control) is disabled; if set to `1`, it is enabled. Default: `1`. Other values are reserved for future extensions. This configuration is applied at startup, after which time the `agc_mode` setting can be changed to `1` or `0` from the main sample application. NOTE: It is not possible to apply different AGC settings for different channels in multi-channel scenarios.

auto_accept_video

Valid Values: `0`, `1`

This setting is only used in auto-answer scenarios when `auto_answer=1`.

If auto_accept_video is set to 1, both audio and video streams are accepted, otherwise incoming calls are answered as audio only, even if video is present in the offer.

auto_accept_video applies to a 3pcc answer when make-callrfc3275 is configured to 1 on the originating DN and a video codec is configured in the endpoint. auto_accept_video is not applied to a 3pcc answer when make-call-rfc3275 is configured to 2 on an originating DN, even if auto_accept_video is set to 1 and a video codec is configured in the endpoint.

auto_answer

Valid Values: `0` or `1`

If set to 1, all incoming calls should be answered automatically.

auto_answer_delay

Valid Values: Number in milliseconds

Default Value: `0`

Time in milliseconds to wait before auto-answering (only applicable when `auto_answer=1` and `auto-answer` is not blocked by missing headset).

callwait_tone_enabled

Valid Values: 0, 1

Default Value: 0

Specifies whether the call waiting tone is enabled for incoming calls, to be played when a new call arrives while user is on active call.

callwait_tone_file

Valid Values: Empty or the path to the call waiting tone sound file. The path can be a file in the current directory or the full path to the sound file.

Default Value: Empty

Specifies the audio file that is played when the call waiting tone is enabled with the `callwait_tone_enabled` option.

The call waiting tone file must be a RIFF (little-endian) WAVE file using one of the following formats:

- kWavFormatPcm = 1, PCM, each sample of size bytes_per_sample
- kWavFormatALaw = 6, 8-bit ITU-T G.711 A-law (8 KHz sampling rate)
- kWavFormatMuLaw = 7, 8-bit ITU-T G.711 mu-law (8 KHz sampling rate)

Uncompressed PCM audio must be 16-bit mono or stereo with a sampling rate of 8, 16, or 32 KHz.

dtmf_method

Valid Values: `Rfc2833`, `Info`, or `InbandRtp`

Method to send DTMF

dtmf_feedback

Valid Values: 0 (disable feedback), 1 (enable feedback)

Default Value: 1

Controls whether local feedback tone is played when Endpoint sends DTMF to remote party.

echo_control

Valid Values: `0` or 1

If set to 1, echo control is enabled.

noise_suppression

Valid Values: `0` or 1

If set to 1, noise suppresion is enabled.

dtx_mode

Valid Values: `0` or 1

If set to 1, DTX is activated.

reject_session_when_headset_na

Valid Values:`0` or 1

If the SDK is configured to use the headset setting (`policy.device.use_headset`=1) and the `reject_session_when_headset_na` option is set to `1`, the SDK should reject the incoming session if a USB headset is not available.

ringing_while_call_held

Valid Values: `0` or `1`

Default Value: `1`
If set to `0`, when another call is held, playing ringtone is suppressed and call wait tone is played instead (if configured). If set to `1` (default value), ringtone is played whenever a new call arrives and there are no other active calls; held calls are not considered active in this case.

sip_code_when_headset_na

Valid Values: Number

Default Value: `480`
If a valid SIP error code is supplied, the SDK rejects the incoming session with the specified SIP error code if a USB headset is not available.

vad_level

Valid values: 0–3,from 0 (conventional VAD) to 3 (aggressive high)

Sets the degree of bandwidth reduction.

ringback_enabled

Valid Values: `0`, `1`, `2`, `3`, `4`, `6`

Default Value: `2`

Specifies whether the ringback tone is enabled for outgoing calls.

- `0` — The ringback is not played when the INVITE dialog is not yet established. In scenarios where ringback is provided by Media Server, the ringback tone would be still present.
- `1` — The incoming media stream is played if provided by the Media gateway in a reliable provisional response with SDP.
- `2` — A local file is used for the ringback.
- `3` — The ringback is always played using either a local file or media provided by the gateway, if the provisional response is reliable.
- `4` — Same as 1, but the incoming media stream is played even if the provisional response from Media gateway is not reliable.
- `6` — The ringback is always played using either a local file or media provided by the gateway (regardless of whether the provisional response is reliable or not).

ringback_file

Valid Values: Empty or the path to the ringback sound file. The path can be a file in the current directory or the full path to the sound file.

Default Value: Empty

Specifies the audio file that is played when the ringing tone is enabled with the `ringing_enabled` option.

WebRTC does not support MP3 playback. The ringtone file for built-in ringback should be a RIFF (little-endian) WAVE file using one of the following formats:

- kWavFormatPcm = 1, PCM, each sample of size bytes_per_sample
- kWavFormatALaw = 6, 8-bit ITU-T G.711 A-law (8 KHz sampling rate)
- kWavFormatMuLaw = 7, 8-bit ITU-T G.711 mu-law (8 KHz sampling rate)

Uncompressed PCM audio must be 16-bit mono or stereo with a sampling rate of 8, 16, or 32 KHz.

ringing_enabled

Valid values: An integer between 0 to 7.
Default Value: 3

- 0—event Ringing disabled.
- 1—event Ringing enabled.
- 2—play ringtone internally (event Ringing disabled).
- 3—play ringtone internally and enable event Ringing.
- 4—play ringtone through a separate ringer device.
- 5—play ringtone through ringer device and enable event Ringing.
- 6—play ringtone internally once for full duration (**ringing_timeout** is not used and the ringing does not stop when call is answered).
- 7—play ringtone once for full duration and enable event Ringing.

Important

If the ringtone is configured to play for full duration (values *6* or *7*), then it will be played even in case of auto-answer with zero delay (configured either in SIP Endpoint SDK or desktop application), unless it is suppressed. This configuration setting assumes **ringing_file** option being configured with a short "beep" sound that would not interfere much with agent hearing the customer (as the ringtone will be played in parallel / mixed with remote audio stream).

Suppressing the Ringtone

The ringtone is generated for all incoming calls to the Genesys SIP Endpoint SDK. To suppress the ringtone for third-party call control for the originating DN, configure the following SIP Server option:

- `make-call-alert-info=<urn:alert:service:3pcc@genesys>`

or

- `make-call-alert-info=<file://null>;service=3pcc`

Important

If at least one application based on SIP Endpoint SDK in the contact center is configured with the `ringing_enabled` option set to a non-zero value, the SIP Server `make-call-alert-info` option should be set to one of the specified values.

ringing_timeout

Valid Values: Empty, 0, or a positive number
Default Value: 0

Specifies the duration, in seconds, of the ringing tone. If set to `0` or if the value is empty, the ringing time is unlimited.

ringing_file

Valid Values: Empty string or string to the path to the ringing sound file. The path may be a file name in the current directory or the full path to the sound file.
Default Value: `ringing.wav`

Specifies the audio file that is played when the ringing tone is enabled with the `ringing_enabled` option.
Note that WebRTC does not support MP3 playback. The ringtone file for built-in ringing should be a RIFF (little-endian) WAVE file using one of the following formats:

kWavFormatPcm = 1, PCM, each sample of size bytes_per_sample
kWavFormatALaw = 6, 8-bit ITU-T G.711 A-law
kWavFormatMuLaw = 7, 8-bit ITU-T G.711 mu-law

Uncompressed PCM audio must be 16 bit mono or stereo and have a frequency of 8, 16, or 32 KHZ.

reject_session_when_busy

Valid Values: Empty, `0`, 1

Default Value: `0`

- `0` or Empty—disable rejection of a session when busy
- 1—enable rejection of a session when busy

use headset

number_sessions_for_busy

Valid Values: Positive integer

Default Value: 1

Sets the number of sessions before busy. Must be a positive integer.

sip_code_when_busy

Valid Values: Empty, 4xx, 5xx, 6xx

Default value: Empty

SIP error response code to use when busy. Can be set to any valid SIP error response code in the 4xx, 5xx, or 6xx range, for example, 486.

rx_agc_mode

Valid Values: `0`, 1

Default value: `0`

When set to 1, the SDK enables the receiving-side AGC allowing the volume of the received RTP stream to be adjusted automatically. When set to `0` (default), the feature is disabled.

`device` Section

audio_in_device

Valid Values: A regex that matches the ECMAScript standard.

Microphone device name. For more information, see Audio Device Settings

audio_out_device

Valid Values: A regex that matches the ECMAScript standard.

Speaker device name

capture_device

Valid Values: A regex that matches the ECMAScript standard.

Capture device name.

ringer_device

Valid Values: A regex that matches the ECMAScript standard.

Name of the device dedicated to playing ringing tone (used when `ringing_enabled` set to 4 or 5)

headset_name

Valid Values: A regex that matches the ECMAScript standard.

The name of the headset model.

exclude_headset

Valid Values: A regex that matches the ECMAScript standard.

Default Value: Empty

When set, this will exclude devices whose name matches the regex pattern from being considered a valid headset for automatic device selection. Microphone and speaker parts of the excluded headset can still be selected manually (or even automatically, if no better device is found), but such a selection will not set "headset available" flag.

use_headset

Valid values: `0` or `1`

If set to `0`, the audio devices specified in `audio_in_device` and `audio_out_device` are used by the SDK. If set to `1`, the SDK uses a headset as the preferred audio input and output device and the audio devices specified in `audio_in_device` and `audio_out_device` are ignored.

include_headset

Valid Values: Pair of device names or name parts with microphone and speaker names separated by colon, or comma-separated list of such pairs, for example: `External Mic : Headphones`

Default Value: Empty

When set, specifies the list of audio in/out devices to be considered as headset for automatic device selection, applicable to case when *use_headset = 1*. The names including delimiter character (quotes, colon or comma) must be enclosed in single or double quotes.

`codecs` Section

See Working with Codec Priorities.

`proxies` Domain

`proxy <n>` Section

display_name

Valid Values: String

Proxy display name

domain

Valid Values: String containing any valid SIP domain
Default Value: Empty string

A SIP domain is an application layer configuration defining the management domain of a SIP proxy. The configured value should include `hostport` and may include `uri-parameters` as defined by RFC 3261. The scheme, `userinfo`, and `transport` URI parameters are included automatically.

If set to an empty string, SIP Endpoint SDK uses the parameters from the `Connectivity` section to construct the SIP domain value as it did in previous versions.

password

Valid Values: String

Proxy password. Password configured for DN object.

reg_interval

Valid Values: Integers greater than or equal to `0`
Default Value: `0`

The period, in seconds, after which the endpoint starts a new registration cycle when a SIP proxy is down. Valid values are integers greater than or equal to `0`. If the setting is empty or negative, the default value is `0`, which means no new registration cycle is allowed. If the setting is greater than `0`, a new registration cycle is allowed and will start after the period specified by `regInterval`.

reg_match_received_rport

Valid Values: `0` or 1
Default Value: `0`

This setting controls whether or not SIP Endpoint SDK should re-register itself when receiving a mismatched IP address in the `received` parameter of a `REGISTER` response. This helps resolve the case where SIP Endpoint SDK for OS X has multiple network interfaces and obtains the wrong local IP address. A value of 0 (default) disables this feature and a value of 1 enables re-registration.

reg_timeout

Valid Values: Number

The period, in seconds, after which registration should expire. A new `REGISTER` request will be sent before expiration. Valid values are integers greater than or equal to `0`. If the setting is `empty/null`, then registration is disabled, putting the endpoint in standalone mode.

`mailbox` Section

Important
The **mailbox** section is a sub-section of the **proxy<n>** section.

password

Valid Values: String

Mailbox password

server

Valid Values: String

Proxy server address and port for this mailbox

timeout

Valid Values: Any positive integer
Default Value: `1800` seconds (30 minutes)

Subscription expiration timeout in seconds. If the setting is missing or set to `0`, the SDK uses a default timeout of 1800 seconds (30 minutes).

transport

Valid Values: `udp`, `tcp`, or `tls`

Transport protocol to use when communicating with server

user

Valid Values: String

Mailbox ID for this mailbox

nat Sub-section

Important

nat is a sub-section of the **proxy<n>** section.

ice_enabled

Valid Values: Boolean

Enable or disable ICE.

stun_server

Valid Values: String

STUN server address. An empty or null value indicates this feature is not used.

stun_server_port

Valid Values: Valid port number

Default Value: 3478

STUN server port value.

turn_password

Valid Values: String

Password for TURN authentication.

Warning

Starting from 9.0.012.02, this setting is deprecated and is not recommended for use, unless suggested by Genesys Technical Support to fix specific problems. Use the GCTI_TURN_PASSWORD environment variable to set the password for TURN authentication.

turn_relay_type

Valid Values: `0`, udp, `1`, or `tcp`

Type of TURN relay.

- `0` or udp for TURN over UDP.
- `1` or `tcp` for TURN over TCP.

turn_server

Valid Values: String

TURN server address. An empty or null value indicates this feature is not used.

turn_server_port

Valid Values:Valid port number

Default Value: 3478

TURN server port value.

turn_user_name

Valid Values: String

User ID for TURN authorization

Warning
Starting from 9.0.012.02, this setting is deprecated and is not recommended for use, unless suggested by Genesys Technical Support to fix specific problems. Use the GCTI_TURN_USERNAME environment variable to set the username for TURN authentication.

`system` Domain

`diagnostics` Section

enable_logging

Valid Values: `0` or `1`

Valid values of `0` or `1` disable or enable logging.

log_file

Valid values: String containing full path or relative path to the log file.

Log file name, for example, `SipEndpoint.log`

log_filter

Valid Values: *Empty*, `dtmf`

Default Value: *Empty*

Specifies the list of log filters to be applied to hide sensitive data from the endpoint log. Currently the only supported filter is `dtmf`, which hides all occurrences of DTMF data from the log (by replacing entered digits with 'x').

log_level

Valid Values: Number, `0–4`

Valid values correspond to log levels:

- 0 = "Fatal"
- 1 = "Error"
- 2 = "Warning"
- 3 = "Info"
- 4 = "Debug".

log_options_provider

Valid Values: Comma-separated list of log setting for various low-level components, including:

- `gsip=N` - Genesys SIP library log level, default is 2
- `webrtc=(level)` - log level(s) for third-party WebRTC native code component. The default is `error` to include error messages and important WebRTC diagnostic, value `api` adds low-level API printouts.

Example value: `gsip=2,webrtc=(error)`

Warning

This settings control low-level debug information used for troubleshooting. Please don't change the value unless instructed by Genesys Technical Support engineer.

logger_type

Valid Values: `external`

If set to `external`, an external logger is used.

`security` Section

Important

SIP Endpoint SDK no longer uses the **tls_enabled** setting.

certificate

See Configuring certificate option for TLS.

certificate-key

See Configuring certificate-key option for TLS.

trusted-ca

See Configuring trusted-ca option for TLS.

sec-protocol

See Configuring sec-protocol option for TLS.

cipher-list

See Configuring cipher-list option for TLS.

ciphersuites

See Configuring ciphersuites option for TLS.

tls-crl

See Configuring tls-crl option for TLS.

tls-target-name-check

See Configuring tls-target-name-check option for TLS.

use_srtp

Valid Values: `optional`, `allowed`, `disabled`, `off`, `elective`, `both`, `enabled`, `force`, `mandatory`

Indicates whether to use SRTP:

- `optional` or `allowed`—do not send secure offers, but accept them
- `disabled` or `off`—do not send secure offers and reject incoming secure offers
- `elective` or `both`—send both secure and non-secure offers and accept either
- `enabled`—send secure offers, accept both secure and non-secure offers
- `force` or `mandatory`—send secure offers, reject incoming non-secure offers

Adding either `',UNENCRYPTED_SRTCP'` (long form) or `',UEC'` (short form) to any value (for example, "*enabled,UEC*`""`), would result in the **UNENCRYPTED_SRTCP** parameter being added to that offer. When this parameter is negotiated, RTCP packets are not encrypted, but are still authenticated.

Producing RTCP Extended Reports

You can use SIP Endpoint SDK to produce RTCP Extended Reports (RFC 3611) and publish them according to RFC 6035 at the end of each call, using a collector address of your choice.

Important
The publish message is sent to the specified collector address only if the `vq_report_collector` parameter is configured with the

`user@server:port;transport=udp` format. For example, `collector@127.0.0.1:5160;transport=udp`.

Settings:

```
<domain name="policy">
<section name="endpoint">
...
<!--
Valid values for Voice Quality (VQ) report publish setting (vq_report_publish):
0--VQ report is not published
1--VQ report is published to the collector at the end of the call--
  see the vq_report_collector setting information below
-->
<setting name="vq_report_publish" value="0"/>
<!--
Valid values for Voice Quality (VQ) report collector setting (vq_report_collector):
NULL or Empty--The VQ report is published to the proxy described in the
Connectivity section
FQDN or IP address along with port and transport--
 collector@SipServer.genesyslab.com:5060;transport=udp
-->
<setting name="vq_report_collector"
value="collector@SipServer.genesyslab.com:5060;transport=udp"/>
</section>
```

Endpoint:

The `vq_report_publish` and `vq_report_collector` settings can be read from the Endpoint Policy by using the following methods:

```
GetEndpointPolicy(EndpointPolicyQuery.VqReportCollector);
GetEndpointPolicy(EndpointPolicyQuery.VqReportPublish);
```

Working with Codec Priorities

Codecs are listed by name in the codecs domain of the configuration file. The **enabled** section explicitly defines the enabled codecs and their priorities. If the **enabled** section is not included or if the audio and video settings have empty values, then the priority is defined as the same order as listed in the codec setting. To disable a codec, remove the codec name from the list.

Important

Media connections between two SIP endpoints (or between SIP Endpoint and Media Server/Gateway) can be successful only when they have a codec that they both support. In tightly controlled environments, it may be possible to limit the number of enabled codecs to only one preferred codec, but it is highly recommended to include the fallback codec(s) like universally supported PCMU and PCMA to the end of the priority list.

The `audio` setting can include a list of comma-separated audio codec names in the order of their priorities. The `video` setting can include a list of comma-separated video codec names in the order of their priorities.

Codec parameters can be specified inline as a comma-separated list enclosed in parenthesis after an equal (=) sign.

```
<setting name="audio" value="g729=(fmtp='annexb=no'),opus=(pt=125),pcmu,pcma"/>
```

The `maxpkt` optional video parameter specifies the maximum RTP packet size for video codecs (including the RTP header but not counting the UDP and IP overheads). The valid values for this parameter are integers from 1024 to 1472, and the default value is 1442. (Any number outside the valid range would be adjusted to the closest allowable value.)

```
  <setting name="video" value="vp8,maxpkt=1280"/>
```

The setting `payload_type` is an integer. It should be specified for codecs with dynamic payload type, such as iSAC, iLBC, OPUS, and the valid values are between 96 and 127.

The setting `fmtp` is a string with valid values as defined by RFC3555 for codec *g729* and RFC7587 for codec *Opus*'.

Important

While any parameter defined by RFC7587 can be specified as the value of the `fmtp` for the Opus codec, SIP Endpoint SDK considers only the `maxaveragebitrate` parameter, which specifies the maximum average bit rate of a session in bits per second (bit/s). Bit rate values can be from *6000* to *64000* (any value outside the range will be adjusted to the minimum/maximum allowed). Recommended bit-rate values:

- from 6000 to 8000 to save bandwidth, with voice quality comparable to G.729 codec.
- 10000 for narrow-band quality (comparable to G.711).
- 20000 for wide-band quality.

The codec *ulpfec/90000* supports the *vp8* codec with forward error correction.

Example

```
<domain name="codecs">
   <section name="enabled">
    <setting name="audio" value="g729,pcmu,pcma"/>
    <setting name="video" value="h264,maxpkt=1200"/>
  </section>
   <section name="G729/8000">
    <setting name="fmtp" value="annexb=no"/>
   </section>
  <section name="PCMU/8000"/>
  <section name="PCMA/8000"/>
  <section name="G722/16000"/>
  <section name="iLBC/8000">
    <setting name="payload_type" value="102"/>
  </section>
  <section name="iSAC/32000">
    <setting name="payload_type" value="104"/>
  </section>
  <section name="iSAC/16000">
    <setting name="payload_type" value="103"/>
  </section>
  <section name="g729/8000">
    <setting name="fmtp" value="annexb=yes"/>
  </section>
  <section name="opus/48000/2">
    <setting name="payload_type" value="120"/>
    <setting name="fmtp" value="maxaveragebitrate=20000"/>
  </section>
  </section>
 <section name="vp8">
    <setting name="payload_type" value="100"/>
  </section>
  <section name="vp9">
     <setting name="payload_type" value="101"/>
  </section>
  <section name="h264">
    <setting name="payload_type" value="108"/>
    <setting name="fmtp" value="profile-level-id=420028"/>
  </section>
  <section name="ulpfec/90000">
    <setting name="payload_type" value="97"/>
  </section>
</domain>
```

Configuring Capture Devices

In order to define the list of preferred capture devices and their priorities, the devices should be added to the device policy section in the **SipEndpoint.config** file:

```
      <domain name="policy">
        <section name="device">
        <!--
          The priority of a device depends on its position in this section.
          The higher the position, the higher device priority.
          To disable a device it should be commented out or removed from the file.
        -->
. . .
          <!-- Capture -->
          <setting name="capture_device" value="CaptureDeviceName-HighPriority"/>
          <setting name="capture_device" value="CaptureDeviceName-LowPriority"/>
        </section>
      </domain>
```

Audio Device Settings

You can configure a headset and other audio input devices using the following parameters in the configuration file:

- `use_headset`
- `headset_name`
- `exclude_headset`
- `audio_in_device`
- `audio_out_device`

If the SDK cannot access any of the configured audio devices, the SDK uses the default system audio devices.

If `use_headset` is set to 1, the SDK searches a headset with name matching configured `headset_name` but not matching `exclude_headset` value. If the SDK does not find the headset, it uses the system defaults.

If `use_headset` is set to `0`, the SDK skips the search for a headset and searches for audio in and audio out devices with names configured by `audio_in_device` and `audio_out_device`. If the devices are not found, the SDK uses the system defaults.

The procedure for audio device selection is applied on startup and every time any changes are made to device presence (such as when a new device is plugged in or an existing device is removed)

Important
A device is selected when the device name matches or partially-matches the regular expression configured in the parameters.

Auto-answer and Call Rejection

The auto-answer and call rejection features depend on the `use_headset`, `auto_answer`, and `reject_session_when_headset_na` configuration settings and whether or not audio devices are available. The following tables describe auto-answer and call rejection behavior:

Auto-answer and Call Rejection when use_headset=1

Headset is Available The SDK considers a headset available if the headset is found by name. Outgoing calls can be initiated.	`auto_answer=1`	Incoming calls are answered automatically: • As audio if `auto_accept_video=0` • As audio with video if the call has video and `auto_accept_video=1`
	`auto_answer=0`	Incoming calls are answered manually and the user explicitly

		selects whether or not video streams should be accepted (using the `has_video` parameter supplied in the `gs_session_info` argument)
Headset is Not Available The SDK decides that no headset is available if a headset was not found by name. An audio device is still assigned, if possible (that is, if any supported devices are present in the system), using the first available audio input and output devices or the system defaults.	No auto-answer is possible in this sub-case, so the `auto_answer` setting is not used	`reject_session_when_headset_na=1` • Incoming calls are automatically rejected • Outgoing calls are blocked
		`reject_session_when_headset_na=0` • Incoming calls can be answered manually—it is assumed that the agent will plug the headset in (or use an available non-headset device, if applicable) before answering the call • Outgoing calls can be initiated—it is the agent's responsibility to ensure that the appropriate audio devices are available before the call is answered by the remote side

Auto-answer and Call Rejection when use_headset=0

Audio devices are configured using the `audio_in_device` and `audio_out_device` settings. If no valid `audio_in_device` and `audio_out_device` devices are found, the SDK selects the system defaults. Outgoing calls can be initiated.

Both microphone and speaker are available	`auto_answer=1`	Incoming calls are answered automatically: • As audio if `auto_accept_video=0` • As audio with video if the call has video and `auto_accept_video=1`
	`auto_answer=0`	Incoming calls are answered manually and the user explicitly selects whether or not video streams should be accepted (using the `has_video` parameter supplied in the `gs_session_info` argument)
Either microphone or speaker is not available	No auto-answer is possible in this sub-case, so the `auto_answer` setting is not used	Auto-rejection is not applicable, so the `reject_session_when_headset_na`

• Incoming calls can be answered manually—it is assumed that the agent will plug in the headset (or use an available non-headset device, if applicable) before answering the call • Outgoing calls can be initiated—it is the agent's responsibility to ensure that the appropriate audio devices are available before the call is answered by the remote side		setting is not used

Rejecting A Call

For backward compatibility with previous releases, a call can only be rejected when both of the following conditions are met (a policy of `should answer` returns `false`):

- Both `use_headset` and `reject_session_when_headset_na` are set to `1`
- None of the devices listed in the `headset_name` settings is currently present

When these conditions are met, an incoming call is rejected with a SIP response code as configured in the `sip_code_when_headset_na` setting.

SIP Endpoint SDK for OS X QuickStart Application

Important

The following features are not supported:

- Video
- IPv6
- NAT, ICE, TURN, STUN
- SIP Cluster
- SIP Proxy

The easiest way to start using the SIP Endpoint SDK for OS X is with the bundled QuickStart application. This application ships in the same folder as the SDK and is supplied as both a double-clickable application in the `/Bin` folder and as source code in the form of an Xcode project located in the `QuickStart/Src` folder.

Running the QuickStart Application

You can try out the QuickStart application by running `/Bin/QuickStart.app`:

1. Set your configuration settings by editing `/Bin/SipEndpoint.config`.
2. Open and run `/Bin/QuickStart.app`.

Rebuilding the QuickStart Application from the Xcode Project

You can modify the QuickStart application by opening the QuickStart Xcode project, `/QuickStart/Src/QuickStart.xcodeproj`.

Before running your modified QuickStart application, you need to rebuild it in Xcode.

At that point, you can either run the application from within Xcode or you can double-click the application that is contained in the `/Bin` folder. When you rebuild the QuickStart project, the `/Bin/Quickstart.app` file is replaced with your modified QuickStart application.

Supplement to SIP Endpoint SDK

This supplement provides descriptions of new features introduced in SIP Endpoint SDK.

9.0.0 Features Support

The following features are described in this supplement:

Released in Version	Feature Name/Article
9.0.008.00	Support real-time access to audio stream
9.0.008.04	SIP Monitoring
9.0.011.06	Separate Ringer Volume Control
9.0.006.10	Support for Dynamic Reconfiguration
9.0.006.10	Support for Audio Statistics and MOS Calculation
9.0.020.02	ReportSDKOperationalData

Support real-time access to audio stream

Important

This feature was introduced as part of the 9.0.008.04 release.

This feature provides a real-time access to audio stream to an application code, allowing the application to monitor raw audio frames coming from microphone and/or sent to speaker device, and to implement custom processing of these frames (e.g. to add real-time transcription).

OSX

The audio stream real time access is supported by utilizing new method to enable/disable audio monitoring of particular stream direction, and notification delegate that takes an audio frame data object holding specific information about that frame.

```
@protocol GSEndpoint <NSObject>
  /**
   Enable audio processing support for requested stream type
  @param streamType values: 0 - disable support; 1 - mic stream; 2 - speaker stream; 3 - both
streams;
  @returns result of the operation
  */
  - (GSStatus) enableAudioMonitor:(int) streamType;

  @protocol GSEndpointNotificationDelegate <NSObject>
  /**
  Called when an audio frame received.
  @see GSEndpointEvent
  */
  - (void) audioFrameReceivedNotification:(GSAudioFrame*) audioFrame;
  /**
  Audio frame data structure supplied with the notification delegate.
  @field direction to indicate the collected media steam type: mic or speaker
  @field samples to hold an array of collected media samples in the received frame
  @field length to hold the count of the stored in array samples
  @field samplingFrequency to hold a frequency
  @field isStereo to indicate whether the the received frame content has stereo or mono data
  @see GSEndpointEvent
  */
  @interface GSAudioFrame : NSObject {
    @private
    int direction;
    NSArray *samples;
    int length;
    int samplingFrequency;
    bool isStereo;
  }
```

.NET

The audio stream real time access support is added to the IExtendedService interface:

```
// Audio related
  //0 - processing disabled; 1 - mic stream; 2 - speaker stream; 3 - both streams;
  GsStatus EnableAudioMonitor(int streamType);
  event EventHandler<EndpointEventArgs^>^ AudioFrameDelivered;
```

Audio frame data is incorporated into endpoint event property dictionary,

```
  EndpointEventArgs^ event;
  IDictionary<String^, Object^>^ property = event->Context->Properties;
```

where the property will hold audio frame data as key value pairs:

```
("direction", direction);                  /* int */
("samples", samples[length]);              /* int16_t samples[] */
("length", length);                        /* int  */
("samplingFrequency", samplingFrequency);  /* int  */
("isStereo", isStereo);                    /* bool */
```

Detailed Description

When Audio monitoring is enabled for particular direction, it is applicable for current and all future session, until explicitly turned off. Parameter `streamType` is basically a bit mask specifying monitoring state of capture (least significant bit) and playback devices (second bit), with `bit=1` enabling monitoring and `bit=0` disabling it.

One single notification callback is used for both audio streams, with

- `direction` property indicating stream type - 1 for capture, 2 for playback stream
- `samples` array holding the audio data, total of `length` 16-bit signed integers representing PCM samples; when `isStereo` is true, data is in interleaved stereo format: L0,R0,L1,R1,...
- `samplingFrequency` indicating number of samples per second

For narrow-band codecs such as G.711 or G.729, sampling frequency is always 8000 and `isStereo` = false. For wide-band codecs, sampling rate depends upon device capabilities and codec used. Namely:

- sampling rate of captured stream is the maximum rate both codec and device supports, stereo is used only for Opus codec, if the microphone device supports it
- sampling rate and stereo status for playback stream always follows the codec parameters (the rate will be as high as 48000 for Opus codec, with `isStereo` = true)

Important

To provide most flexibility and avoid potential loss of data, SDK never tries to convert audio data it gets from voice engine from one format to another. Application code should implement resampling itself, if needed.

Code Samples

OSX

The OSX code sample is written in the assumption that application code uses class with 'ep' property referring to GSEndpoint object, and a field to keep observed statistics. That is, it includes the following in the app header:

```
@property (nonatomic, retain) id<GSEndpoint> ep;
@property int frequencyMicAvg;
```

and the following initialization code:

```
self.ep = [GSEndpointFactory initSipEndpoint:configData];
frequencyMicAvg = 0;
```

Start microphone monitoring

Method to start monitoring of microphone input audio stream:

```
- (void) startMicMonitor
{
  frequencyMicAvg = 0;
  GSStatus result = [self.ep enableAudioMonitor:1];
  NSLog(@"Start mic stream audio monitor result:%d", result);
}
```

Handle audio frame data

Sample audio monitoring method approximate the main frequency of the audio frame (by calculating number of zero crossings) and exponential moving average (EMA) of that frequency for the entire duration of monitoring. Note that, because each frame consists of 10 msec of audio samples, sampling frequency is not actually used in calculations.

```
- (void) audioFrameReceivedNotification:(GSAudioFrame *)audioFrame
{
  int len = audioFrame.length;
  int sfq = audioFrame.samplingFrequency;
  int di  = audioFrame.isStereo ? 2 : 1;
  int zxs = 0;
  for (int i = 0; i < len-di; i += di)
    if (([[audioFrame.samples objectAtIndex: i]    intValue] > 0) !=
         [[audioFrame.samples objectAtIndex:(i+1)] intValue] > 0)) zxs++;
  float f = (zxs / 2) * 100;
  if  (frequencyMicAvg == 0) frequencyMicAvg = f;
  else frequencyMicAvg =  (9*frequencyMicAvg + f)/10;
}
```

Finish monitoring

Method to stop monitoring and print accumulated statistic:

```
  - (void) stopMicMonitor
  {
    [self.ep enableAudioMonitor:0];
    NSLog(@"frequencyMicAvg:%d", frequencyMicAvg);
  }
```

.NET

The .NET code sample is written in the assumption that the application code uses class with `endpoint` property, and a field to keep observed statistics:

```
  private SipEndpoint.IEndpoint endpoint;
  float frequencyMicAvg;
```

and the following initialization code:

```
  this.endpoint = SipEndpoint.EndpointFactory.CreateSipEndpoint();
  this.extendedService = this.endpoint.Resolve("IExtendedService") as ExtendedService;
```

Start microphone monitoring

Method to start monitoring of microphone input audio stream:

```
  GsStatus StartMicMonitor(int direction)
  {
    frequencyMicAvg = -1.f;
    return this.extendedService.EnableAudioMonitor(1); // DeviceTypeMicrophone=1
  }
```

Handle audio frame data

Sample audio monitoring method approximate the main frequency of the audio frame (by calculating number of zero crossings) and exponential moving average (EMA) of that frequency for the entire duration of monitoring. Note that, because each frame consists of 10 msec of audio samples, sampling frequency is not actually used in calculations.

```
  void OnAudioFrameDelivered(object sender, EndpointEventArgs e)
  {
    int len = (int)e.Context.Properties["length"];
    int sfq = (int)e.Context.Properties["samplingFrequency"];
    int di  = (int)e.Context.Properties["isStereo"] ? 2 : 1;
    int[] s = (int[])e.Context.Properties["samples"];
    int zxs = 0;
    for (int i = 0; i < len-di; i += di)
      if ((s[i] > 0) != (s[i+di] > 0)) zxs++;
    float f = (zxs / 2) * 100;
    if  (frequencyMicAvg < 0) frequencyMicAvg = f;
    else frequencyMicAvg = (9*frequencyMicAvg + f)/10;
  }
```

Finish monitoring

Method to stop monitoring and print accumulated statistic:

```
  void StopMicMonitor()
  {
    this.extendedService.EnableAudioMonitor(0);
    this.extendedService.LogPrint(4,"frequencyMicAvg:" + frequencyMicAvg.ToString());
  }
```

Separate Ringer Volume Control

SEPSDK-2261

We are sorry. The content of this JIRA could not be loaded.

Important

This feature was introduced as part of the 9.0.011.06 release.

This feature allows the application to control ringer volume without affecting the actual output device volume, by scaling the ringtone audio before playing it to the device. The implementation provides both **set** and **get** methods, and a method to test current volume by playing a fragment of the configured ringtone.

OS X

The ringer control functionality is included a part of the **GSEndpoint** interface.

```
  @protocol GSEndpoint <NSObject>
  /**
   Ringer volume scaling, a number between 0 and 100 representing the current scaling,
   with 0 muting the ringtone, 50 meaning "no scaling" (no changes to ringtone volume)
   and 100 meaning "maximum amplification" (volume increased to 2 times louder)
   */
  @property (nonatomic) int ringerVolume;
  /**
   Changes the ringer volume scaling to the specified value.
   @param value a number between 0 and 100 representing the new scaling.
   @returns the result of the operation
   */
  - (GSResult) changeRingerVolumeTo:(int) value;
  /**
   Get the currently used ringer volume.
   @return a number between 0 and 100 representing the current scaling
   */
  - (int) getRingerVolume;
  /**
   Plays configured ringing tone for specified duration (to test volume).
   @param duration of playback in milliseconds.
   @returns the result of the operation
   */
- (GSResult) testRingerPlayback:(int) value;
```

.NET

The ringer volume control support has been added as the speaker related method of the **IExtendedService** interface.

```
public interface class IExtendedService
  {
    GsStatus SetRingerVolume(int volume);
    int      GetRingerVolume();
    GsStatus TestRingerPlayback(int duration_msec);
  }
```

Detailed Description

Similar to the session-level speaker volume control, the value of ringer volume scaling should be an integer from 0 (effectively muting the sound) to 100 (maximum amplification). However, for ringer volume, the maximum amplification is different, it is only 2 times louder than the original waveform.

Because of the limitation of Voice Engine being used, the scaling factor for file playback is set at the beginning and cannot be changed during the playback, so a new value from **SetRingerVolume** / **changeRingerVolumeTo** methods is applied only to the next ringing event. In order to make ringer volume adjustment easier, a new method has been added to test the sound.

The **TestRingerPlayback** method may be called at any time, but it will play the sound only when output device is not busy already (i.e., no active session exists and no ringtone is currently playing), otherwise it will take no effect. The file configured in `policy.session.ringing_file` is played from the beginning up to the specified duration.

> **Important**
> In case when the waveform in the configured "ringing" file is already at / near the maximum amplitude (as in the sample provided by Genesys), increasing the audio volume before sending it to the device is technically not possible and would result in a waveform distortion (because standard audio API on all platforms only accepts 16-bit integers, there is a natural limit on how big the amplitude might be). However, currently the SDK does not prevent application / user to scale the waveform past that max-amplitude point, not only to simplify the implementation, but also because - for most common ringtones - the distorted waveform is perceived by human ear as being louder (because wrapping amplified values into 16-bit field effectively results in multiplying the main tone frequency, and higher frequencies sounds "louder").

Code Samples

OS X

The OSX code sample is written in the assumption that application code uses class with 'ep' property referring to GSEndpoint object. That is, it includes the following in the app header:

```
 @property (nonatomic, retain) id<GSEndpoint> ep;
```

and the following initialization code:

```
 self.ep = [GSEndpointFactory initSipEndpoint:configData];
```

The following method gets the current ringer volume, sets the new volume (to value passed as argument), prints what was changed to the log, and plays test sound for 1 second:

```
  - (void) setAndTestRingerVolume:(int) newRingerVolume
  {
    int oldRingerVolume = [self.ep getRingerVolume];
    GSResult result = [self.ep changeRingerVolumeTo:newRingerVolume];
    if (result == GSResultOK) {
      NSLog(@"Ringer volume changed from %@ to %@", oldRingerVolume, newRingerVolume);
      [self.ep testRingerPlayback:1000];
    }
    else NSLog(@"Could not set Ringer volume, rc:%d", result);
  }
```

.NET

The .NET code sample is written in the assumption that the application code uses class with 'endpoint' property:

```
  private SipEndpoint.IEndpoint endpoint;
```

and the following initialization code:

```
this.endpoint = SipEndpoint.EndpointFactory.CreateSipEndpoint();
  this.extendedService = this.endpoint.Resolve("IExtendedService") as ExtendedService;
```

The following method gets the current ringer volume, sets the new volume (to value passed as argument), prints what was changed to the log, and plays test sound for 1 second:

```
  void SetAndTestRingerVolume (int newRingerVolume)
  {
    int oldRingerVolume = this.extendedService.GetRingerVolume();
    GsStatus result = this.extendedService.SetRingerVolume(newRingerVolume);
    if (result == GsStatusSuccess) {
      this.extendedService.LogPrint(4,"Ringer volume changed from
"+oldRingerVolume.ToString()+
                                                              " to
"+newRingerVolume.ToString());
      this.extendedService.TestRingerPlayback(1000);
    }
    else this.extendedService.LogPrint(4,"Could not set Ringer volume,
rc:"+result.ToString());
  }
```

Appendix: Details of waveform scaling implementation

A third-party Voice Engine (taken from Goggle's open source WebRTC Native Code) uses slightly different algorithms for session-level volume and file playback volume scalings, although SIP Endpoint SDK provides the same range for `volume` parameter.

Waveform scaling for session-level volume is done as:

```
double GSeptBase::vol_int0to100_to_scaling_factor (int volume_0_to_100)
{
  double X = (double)volume_0_to_100/50.0 - 1.0; // mapping 0..100 -> 0.0..4.25
  double scaling = pow(4,X) + 0.25*X + 0.000001; // by exponent 50 -> 1.0
  if (scaling <  0.0) scaling =  0.0;
  if (scaling > 10.0) scaling = 10.0; return scaling;
}
int AudioFrameOperations::ScaleWithSat(float scale, AudioFrame& frame)
{
  // Ensure that the output result is saturated [-32768, +32767].
  int32_t temp_data = 0;
  for (int i = 0; i < frame.samples_per_channel * frame.num_channels; i++) {
    temp_data = static_cast<int32_t>(scale * frame.data[i]);
         if (temp_data < -32768)  frame.data[i] = -32768;
    else if (temp_data >  32767)  frame.data[i] =  32767;
    else frame.data[i] = static_cast<int16_t>(temp_data);
  }
  return 0;
}
```

On the other hand, the **FilePlayerImpl::SetAudioScaling()** method only accepts scaling factors from 0 to 2.0, unlike SetChannelOutputVolumeScaling() in VoEVolumeControl interface => use linear mapping, to keep the same 0..100 range, with 50 mapped to 1.0 to provide the default "no ring volume scaling":

```
int GSeptBase::SetRingerVolume(int volume_0_to_100)
{
  ringer_scaling = (double)volume_0_to_100/50.0;
  ...
}
int32_t FilePlayerImpl::Get10msAudioFromFile(int16_t* outBuffer, ...)
{ ...
  if (_scaling != 1.0) {
    for (int i = 0; i < outLen; i++)
      outBuffer[i] = (int16_t)(outBuffer[i] * _scaling);
  }
  return 0;
}
```

SIP Monitoring

Important

This feature was introduced as part of the 9.0.008.04 release.

This feature allows custom application to get SDP content and negotiated codec name for the currently active session, and to get SIP messages for active or recently released session.

OS X

The following methods have been added to the GSSessionService:

```
/**
 Session service protocol
 */
@protocol GSSessionService <NSObject>
...
/**
 Gets codec name for active or held session.
 @param sessionId active or held session ID.
 @returns the code name.
 */
- (NSString*) getCodecNameForSessionId:(int) sessionId;
/**
 Gets last remote offer or answer SDP for active or held session.
 @param sessionId active or held session ID.
 @returns SDP content.
 */
- (NSString*) getRemoteSDPForSessionId:(int) sessionId;
/**
 Gets specific SIP message from history by its position index.
 @param sessionId active or released session ID.
 @param index position in SIP history.
 @returns SIP message content.
 */
- (NSString*) getSIPMessageForSessionId:(int) sessionId
                                 byIndex:(int) index;
/**
 Gets specific SIP message from history by its name.
 @param sessionId active or released session ID.
 @param name SIP message name.
 @returns SIP message content.
 */
- (NSString*) getSIPMessageForSessionId:(int) sessionId
                                  byName:(NSString*) name;
```

.NET

The following methods have been added to the ICallControl interface:

```
public interface class ICallControl {
...
  String^ GetCodecName (int sessionId);
  String^ GetRemoteSDP (int sessionId);
  String^ GetSIPMessage(int sessionId, int index);
  String^ GetSIPMessage(int sessionId, String^ name);
}
```

Detailed Description

All methods gets integer session ID (of the session queried) as the first parameter, which is the same ID as used in all session-related methods and which should be taken from session object by retrieving:

- property "`callId`" in SDK for OS X (note that, for historical reasons, property named "sessionId" holds different value - namely, the value of SIP header Call-ID for given session)
- property "`SessionIntId`" in SDK for .NET

Return value is a string representation of requested value, as described below, or empty string if the operation cannot be performed of requested data is not available.

Getting codec name

Corresponding method returns canonical name of currently negotiated codec for given session (e.g. "pcmu/8000" or "opus/48000/2")

Getting remote SDP content

Corresponding method returns last remote SDP received for given session as text

Getting SIP messages

Two methods are provided to obtain specific SIP message for given session, using either:

- message index (starting from 1) in the sequence of all SIP messages sent or received for this session,
- SIP message name, to get the last SIP message for this session with matching name

For the second approach, name parameter may be either SIP method (e.g. "INVITE" or "BYE"), SIP response code as string (e.g. "200") or special name "1xx" that matches any provisional response. In addition, the value may be prefixed with "<<" (to consider only incoming message) or "">>" (for outgoing messages only). For example:

- "`INVITE`" = last INVITE message (sent or received)

- "<<INVITE" = last incoming INVITE
- ">>ACK" = last ACK sent by endpoint

Return value includes full text of SIP message, including all headers and message body, or empty string if the requested message cannot be found.

Code Samples

OS X

The OSX code sample is written in the assumption that application code uses class with 'ep' property referring to GSEndpoint object. That is, it includes the following in the app header:

```
 @property (nonatomic, retain) id<GSEndpoint> ep;
```

and the following initialization code:

```
  self.ep = [GSEndpointFactory initSipEndpoint:configData];
```

Copying all SIP messages for given session into string array

This code fragment copies all SIP messages for session with given ID of callId into string array:

```
  NSMutableArray *sipMessages = [[[NSMutableArray alloc] init] autorelease];
  for (int i = 1;; i++) {
    NSString *msg = [[self.ep sessionControlService] getSIPMessageForSessionId:callId
byIndex:i];
    if (msg.length == 0) break;
    [sipMessages addObject:msg];
  }
```

Important
Index starts at 1 (not 0) and the condition for end of list is returning empty value.

Printing session info to log

The following code prints some session information into log, including codec name, remote SDP, and "last incoming INVITE":

```
  NSLog(@"Negotiated codec is %@", [[self.ep sessionControlService]
getCodecNameForSessionId:callId]);
  NSLog(@"Remote SDP is %@",       [[self.ep sessionControlService]
getRemoteSDPForSessionId:callId]);
  NSLog(@"Last incoming INVITE is %@", [[self.ep sessionControlService]
                                                getSIPMessageForSessionId:callId
byName:@"<<INVITE"]);
```

.NET

The .NET code sample is written in the assumption that the application code uses class with '`endpoint`' property:

```
  private SipEndpoint.IEndpoint endpoint;
```

and the following initialization code:

```
  this.endpoint = SipEndpoint.EndpointFactory.CreateSipEndpoint();
  this.callControl = this.endpoint.Resolve("ICallControl") as CallControl;
```

Copying all SIP messages for given session into string array

This code fragment copies all SIP messages for session with given ID of callId into string array:

```
  string[] sipMessages = new string[100];
  for (int i = 1; i<100; i++) {
    string msg = this.callControl.GetSIPMessage(callId, i);
    if (msg.Length == 0) break;
    sipMessage[i] = msg;
}
```

Important

Index starts at 1 (not 0) and the condition for end of list is returning empty value.

Printing session info to log

The following code prints some session information into log, including codec name, remote SDP, and "last incoming INVITE":

```
  this.extendedService.LogPrint(4,"Negotiated codec is
"+this.callControl.GetCodecName(callId));
  this.extendedService.LogPrint(4,"Remote SDP is "
+this.callControl.GetRemoteSDP(callId));
  this.extendedService.LogPrint(4,"Last incoming INVITE is "
                                           +this.callControl.GetSIPMessage(callId,
"<<INVITE"));
```

Support for Dynamic Reconfiguration

Important
This feature was introduced as part of the 9.0.006.10 release and updated in 9.0.016.03 release.

This feature provides additional options to support regex matching and specify codec priorities.

This page describes the API and rules for dynamic reconfiguration of SIP Endpoint SDK from an application. Implementation provides two methods: one to get access to current configuration and another to change that configuration dynamically.

OSX

The following methods are defined in the GSEndpoint protocol:

```
@protocol GSEndpoint <NSObject>
/**
 Get endpoint configuration setting
 @param key
 @returns string value of the requested key string
 @returns empty string if key-value pair does not exists
 @returns empty string if key null or empty
 */
- (NSString*) getConfigSettingForKey:(NSString*) key;
/**
 Set endpoint configuration setting
 @param value
 @param key
 @returns result of the operation
 */
- (GSResult) setConfigSettingValue:(NSString*) value forKey:(NSString*) key;
```

.NET

The following methods are defined in the IExtendedService interface:

```
public interface class IExtendedService
{
  void SetConfigStringSetting(String^ key, String^ value);
  String^ GetConfigStringSetting(String^ key);
}
```

Detailed Description

The "key" value (for both configuration methods) should be in either one of these two forms:

1. for reference to Connectivity parameters in "Basic" container -- "*name* :*N*", where
   - *name* is the attribute name, one of `user`, `server`, `protocol`, `transport` (synonym for `protocol`).
   - *N* refers to Connectivity line index, starting from 0, (i.e., the setting "`user:0`" refers to the *user* parameter in the first Connectivity line, and "`user:1`" refers to the second line.)
2. for reference to all setting in Genesys container -- *domain*`.`*section*`.[`*subsection*`.]`*setting*, where
   - *domain* is the XML domain element and must be one of `policy`, `codecs`, `proxies` or `system`
   - *section* and `setting` refer to corresponding XML schema elements
   - *subsection* is optional section name, as used currently for NAT options

Important

Historically, SDK for .NET and OS X used different delimiters in option keys. For backward compatibility, in the current SDK version, dot and colon in the key value may be used interchangeably.

When do the Changes Take Effect?

While any configuration setting may be changed any time, not all changes take effect immediately. Particularly, for most cases:

- Connectivity parameters, settings in `proxies` domain, and `system.security` section take effect when connection is activated,
- Settings in `policy.session` section and in `codecs` domain take effect for the next session created,
- Settings in `policy.device` section take effect next time device is going to be selected.

The following settings in policy.endpoint section that take effect for the next session (without Endpoint restart):

- `include_os_version_in_user_agent_header`
- `include_sdk_version_in_user_agent_header`
- `answer_sdp_priority`
- `defer_device_release`
- `refer_to_proxy`
- `vq_report_publish`
- `vq_alarm_threshold`

The following settings in policy.endpoint section take effect only when connection is activated:

- `public_address`
- `sip_port_binding`

The following settings in policy.endpoint section do not take effect without full SDK or application restart:

- ip_versions
- `sip_port_min`, `sip_port_max`
- `rtp_port_min`, `rtp_port_max`
- `tcp_port_min`, `tcp_port_max`
- `rtp_inactivity_timeout`
- `sip_transaction_timeout`

Code Samples

OSX

The OSX code samples are written in assumption that application code uses class with 'ep' property referring to the GSEndpoint object and it includes the following in the app header:

```
@property(nonatomic, retain)
id <GSEndpoint> ep;
```

and that property is initialized in the app implementation:

```
self.ep = [GSEndpointFactory initSipEndpoint:configData];
```

Changing Session Policy Auto-answer

This example shows how to get the current value and set the policy, `auto-answer` value to *1* (true). Note that policy change will not affect any calls that are already ringing on the endpoint but will take effect from the next call onwards.

```
NSString *oldAA = [self.ep getConfigSettingForKey:"policy.session.auto_answer"];
GSResult result = [self.ep setConfigSettingValue:"1" forKey:"policy.session.auto_answer"];
if (result == GSResultOK)
     NSLog(@"Auto-answer changed from %@ to 1.",oldAA);
else NSLog(@"Error %d changing Auto-answer from %@.", result, oldAA);
```

A similar approach can be used for other settings that do not require the connection or SDK to be restarted to be take effect.

Changing Connectivity Parameters

The following sample shows how to reconnect to a different SIP Server location. To do so, the application should:

- disable a connection with certain `configId` or `connectionId`
- change the server location
- enable the connection again

1. Obtain a connection reference by:
   - `configId` (the position in connections list of the current configuration, starting from *0* index)

     ```
     GSSipConnection *connection = [[self.ep connectionManager]
      connectionByConfigId:configId];
     ```

   - `connectionId` (an ID that is set in the `connectionId` property when enabling the connection)

     ```
     GSSipConnection *connection =
      [[self. ep connectionManager] connectionByConnectionId:connectionId] ;
     ```

2. Changing the server location to `new_server` for given connection:

   ```
   [connection disable]; // that operation cannot fail (always returns GSResultOK)
    NSString *key = [NSString stringWithFormat:@"server:%d", connection.configId];
    GSResult result = [self.ep setConfigSettingValue:new_server forKey:key];
    if (result == GSResultOK) result = [connection enable];
    if (result == GSResultOK)
         NSLog(@"Connection changed to %@.", new_server);
    else NSLog(@"Error %d changing connection.", result);
   ```

Changing Endpoint Setting with Full SDK Restart

The following sample shows how to change a global value (e.g., `policy.endpoint.sip_transaction_timeout`) and then perform a full SDK restart.

```
[self.ep stop] // stop endpoint
  [self.ep setConfigSettingValue:@"5000" forKey:@"policy.endpoint.sip_transaction_timeout"];
if ([self.ep configure]) { // re-start endpoint
  self.ep.notificationDelegate = self;
  self.ep.connectionManager.notificationDelegate = self;
  self.ep.sessionManager.notificationDelegate = self;
  self.ep.deviceManager.notificationDelegate = self;
  [self.ep activate];
  [self.ep.deviceManager configure];
  self.connections = [self.ep.connectionManager allConnections];
}
else NSLog(@"Error restarting");
```

.NET

Code samples in this section are written in assumption that application code uses class with `endpoint` property.

```
 private SipEndpoint.IEndpoint endpoint;
```

and the following initialization code:

```
this.endpoint = SipEndpoint.EndpointFactory.CreateSipEndpoint();
this.extendedService = this.endpoint.Resolve("IExtendedService") as ExtendedService;
this.connectionManager = this.endpoint.Resolve("IConnectionManager") as ConnectionManager;
```

Changing Session Policy Auto-answer

This example shows how to get the current value and set the policy, `auto-answer` value to *1* (true).

> **Important**
> Any change will not affect any calls that are already ringing on the endpoint but will take effect from the next call onwards.

```
String oldAA= this.extendedService.GetConfigStringSetting("policy.session.auto_answer");
GsStatus result = this.extendedService.SetConfigStringSetting("policy.session.auto_answer",
"1");
if (result == GsStatusSuccess)
     this.extendedService.LogPrint(4,"Auto-answer changed from " + oldAA +" to " + newAA);
else this.extendedService.LogPrint(4,"Error" + result + "changing Auto-answer to " + newAA);
```

A similar approach can be used for other settings that do not require the connection or SDK to be restarted to be take effect.

Changing Connectivity Parameters

The following sample shows how to reconnect to a different SIP Server location. To do so, the application should:

- disable a connection with certain `configId` or `connectionId`
- change the server location
- enable the connection again

These steps can be performed as following:

```
this.connectionManager.DisableConnection(connectionId); // that operation cannot fail (always
returns GsStatusSuccess)
String key = "server:" + connection.configId.ToString();
this.extendedService.SetConfigStringSetting(key, new_server);
this.connectionManager.EnableConnection(connectionConfigId);
```

If the application requires the `connectionId` value for a known `connectionConfId`, the following method may be used:

```
 public int GetConnectionIdByConfigId(int connectionConfigId)
 {
   foreach (Connection conn in this.connectionManager.Connections)
   if (conn.ConfId == connectionConfigId) return conn.Id;
   return -1; // negative result means "not found"
 }
```

Changing Endpoint Setting with Full SDK Restart

The following sample shows how to change a global value (e.g., `policy.endpoint.sip_transaction_timeout`) and then perform a full SDK restart.

```
this.extendedService.StopCore(); // stop endpoint
this.extendedService.SetConfigStringSetting("policy.endpoint.sip_transaction_timeout",
"5000");
this.extendedService.RestartCore();
```

Audio statistics and MOS Calculation

This page describes the API to get audio statistics for current or recently completed call, what is included into that statistics, and how the derived metrics (such as MOS) are calculated.

Internal C++ data structure

```
typedef struct gs_call_statistics {
  int got_local_stat; // true if next 5 fields are valid (local stat available)
  float local_fraction_lost; // fraction of lost packets (not locally received)
  int      local_total_lost; // total lost packaged (calculated per RFC 3550)
  unsigned local_jitter;     // interarrival jitter
  unsigned local_PktCount;   // local packet count (number of RTP packets sent)
  unsigned local_OctCount;   // local octet count (number of bytes sent)
  int got_remote_SR; // true if next 2 fields valid (got remote Sender Report)
  unsigned remote_PktCount;  // remote packet count
  unsigned remote_OctCount;  // remote octet count
  int got_remote_RR; // true if next 3 fields valid (got remote Receiver Report)
  float remote_fraction_lost; // fraction of lost pkts (not received by remote)
  int      remote_total_lost; // total lost packaged (calculated per RFC 3550)
  unsigned remote_jitter;     // interarrival jitter
  int rttMs; // Round-trip time in milliseconds.
  float local_MOS;      // local MOS for entire call segment (added in 9.0.005)
  float local_interMOS; // local MOS for the last interval
  //
  // added in 9.0.009 (all decoded_10ms_xxx values are numbers of 10ms frames):
  //
  int got_neteq_stat; // true if next 6 fields are valid (NetEQ stat available)
  int     local_pkt_received; // total number of RTP packets received locally
  pgtime_t last_pkt_received; // timestamp of last RTP or RTCP packet received
  int decoded_10ms_normal;    // AudioDecodingStats - normal (decoded from RTP)
  int decoded_10ms_plc;       // - Packet Loss Concealment
  int decoded_10ms_cng;       // - Comfort Noise Generation
  int decoded_10ms_silence;   // - dead silence
  //
  // information about current (or last used) codec, added in 9.0.012:
  //
  int got_codec_info; // true if next 2 fields valid (codec info is available)
  int         codec_rtp_pt;   // payload type used for sending RTP packets
  const char *codec_sdp_name; // codec name (as appears in SDP)
}
gs_call_statistics;
```

OS X

The following methods are defined in GSSessionService protocol to get audio statistics:

```
  @protocol GSSessionService <NSObject>
  /**
   Get audio statistics for particular session
   @param session object.
   @returns GSStatistics with audio statistics content.
```

```
 */
- (GSStatistics*) audioStatisticsForSession:(id<GSSession>) session;
- (GSStatistics*) audioStatisticsForSessionId:(int) sessionId;
```

where GSStatistics is an object with the following properties (closely matching internal C++ structure, just with names formatted using standard objective-C convention):

```
@property (nonatomic) int gotLocalStat;
@property (nonatomic) float localFractionLost;
@property (nonatomic) int localTotalLost;
@property (nonatomic) unsigned localJitter;
@property (nonatomic) unsigned localPktCount;
@property (nonatomic) unsigned localOctCount;
@property (nonatomic) int gotRemoteSR;
@property (nonatomic) unsigned remotePktCount;
@property (nonatomic) unsigned remoteOctCount;
@property (nonatomic) int gotRemoteRR;
@property (nonatomic) float remoteFractionLost;
@property (nonatomic) int remoteTotalLost;
@property (nonatomic) unsigned remoteJitter;
@property (nonatomic) int rttMs;
@property (nonatomic) float vqLocalMOS;
@property (nonatomic) float vqLocalIntervalMOS;
@property (nonatomic) int gotNeteqStat;
@property (nonatomic) int localPktReceived;
@property (nonatomic) struct timeval lastPktReceived;
@property (nonatomic) int decoded10msNormal;
@property (nonatomic) int decoded10msPlc;
@property (nonatomic) int decoded10msCng;
@property (nonatomic) int decoded10msSilence;
```

.NET

The following method is defined in ICallControl interface to get audio statistics for specific session ID:

```
Dictionary<String^,Object^>^ GetAudioStatistics(int sessionId);
```

where the returned dictionary is created from internal gs_call_statistics structure as following:

```
virtual Dictionary<String^, Object^>^ GetAudioStatistics(int sessionId)
{
  Dictionary<String^,Object^>^ res = gcnew Dictionary<String^,Object^>();
  gs_call_statistics gst = this->endpoint->getAudioStatistics(sessionId);
  res->Add("Got Local Stat ", gst.got_local_stat);
  res->Add("Local Frac Lost", gst.local_fraction_lost);
  res->Add("Local TotalLost", gst.local_total_lost);
  res->Add("Local Jitter   ", gst.local_jitter);
  res->Add("Local Pkt Count", gst.local_PktCount);
  res->Add("Local Oct Count", gst.local_OctCount);
  res->Add("Got Remote SR  ", gst.got_remote_SR);
  res->Add("Remote PktCount", gst.remote_PktCount);
  res->Add("Remote OctCount", gst.remote_OctCount);
  res->Add("Got Remote RR  ", gst.got_remote_RR);
  res->Add("Remote FracLost", gst.remote_fraction_lost);
  res->Add("Remote TotaLost", gst.remote_total_lost);
  res->Add("Remote Jitter  ", gst.remote_jitter);
  res->Add("Round Trip Time",      gst.rttMs);
  res->Add("VQ Local MOS",         gst.local_MOS);
```

```
    res->Add("VQ Local IntervalMOS", gst.local_interMOS);
    res->Add("GotNeteqStat",       gst.got_neteq_stat);
    res->Add("LocalPktReceived",   gst.local_pkt_received);
    res->Add("LastPktReceived",    gst.last_pkt_received);
    res->Add("Decoded10msNormal",  gst.decoded_10ms_normal);
    res->Add("Decoded10msPlc",     gst.decoded_10ms_plc);
    res->Add("Decoded10msCng",     gst.decoded_10ms_cng);
    res->Add("Decoded10msSilence", gst.decoded_10ms_silence);
    return res;
  }
```

Detailed Description

Data returned by SIP Endpoint SDK includes the following raw audio statistics data and derived metrics (refer to following sub-sections for detailed description):

- raw local stats (used to generating RTCP reports),
- remote stats obtained from RTCP reports from other side of conversation,
- estimated MOS values, calculated based on the raw statistics,
- audio decoding statistics from Voice Engine
- current codec info

All locally gathered / calculated statistics are reset each time when RTP session is started (after hold/retrieve or because of codec changed for active session). The SDK always returns statistics for last call segment only, from the moment last RTP session for the call was started.

Raw local stats

This group includes raw local stats that are used for RTCP reports generation, with their meaning defined by RFC 3550, namely:

- `float local_fraction_lost` = same as "fraction lost" in RFC 3550, but expressed as float number
- `int local_total_lost` = "cumulative number of packets lost" in that RFC
- `unsigned local_jitter` = "interarrival jitter"
- `unsigned local_PktCount` = "sender's packet count"
- `unsigned local_OctCount` = "sender's octet count" (for this Endpoint considered a "sender")

These values are always available as soon as RTP transmission starts, right after SDP negotiation is completed and audio devices initialized.

Remote stats from RTCP reports

These two groups include remote-side stats received in RTCP reports, with the same meaning as local stats described in previous section. This information is available only after receiving RTCP packet with Server Report (SR) for remote_Pkt/OctCount values, or Receiver Report (RR) for the other 3 values (for successfully established call, usually RTCP packet contains both reports).

While RFC 3550 provides quite complex rules and algorithms for calculating RTCP transmission interval, most modern implementations (including Geneys SIP Endpoint SDK) use simplified algorithm, sending RTCP with random intervals between 2.5 and 7.5 sec (i.e. on average one RTCP every 5 sec). Since first RCTP packet may not include all data yet, full remote stats are likely to be available only ~10 sec after the media session is started.

Calculated RTT and MOS

This group includes:

- round-trip time (RTT) calculated based on timing date in received RTCP packets, value in milliseconds,
- estimated Mean Opinion Score (MOS) calculated using "E model" from ITU-T recommendation G.107 (see details below), with two value provided:
    - `local_MOS` calculated over the entire duration of RTP session (from the time call was first established, retrieved from hold, or codec was changed),
    - `local_interMOS` calculated for the last RTCP transmission interval (value recalculated every time RTCP is sent)

For all 3 statistics, zero value means "data is not available", i.e. no RTCP received yet to calculate RTT, or not enough incoming RTP packets received to estimate the MOS (current implementation requires at least 5 incoming RTP to start calculations). Refer to the separate section below for the details of MOS calculation.

Audio decoding statistic

This group includes audio processing statistics from Voice Engine network equalizer / decoder:

- total number of RTP packets received locally,
- timestamp of last RTP or RTCP packet received,
- number of 10-msec audio frames that were decoded as:
    - "normal" (i.e. fully from the RTP packet received),
    - affected by Packet Loss Concealment (PLC),
    - added by Comfort Noise Generation (CNG) - only present when remote side uses silence suppression along with Comfort Noise (CN) packets,
    - inserted dead silence.

Because of inevitable delays in network transmission (and two sides not being able to start media stream at exactly the same time), small number of PLC and "silence" frames are fully expected even in perfect conditions.

Codec info

This group includes current (or last used, for inactive session) codec information:

- codec_rtp_pt is RTP payload type as defined in RFC 3550. For most commonly used codec that is a static values assigned by RFC 3551 (that parameter is not very useful for codecs with dynamic payload type):

    - for G.711 - pt=0,"pcmu" or pt=8,"pcma"
    - for G.722 - pt=9,"g722"
    - for G.729 - pt=18,"g729"
- codec_sdp_name is canonical codec name as it appears in SDP, e.g. "opus", "iSAC" or "iLBC" for the rest of audio codecs

MOS calculation in SIP Endpoint SDK

Currently SIP Endpoint SDK uses "E model" from ITU-T recommendation G.107 for MOS calculation, with codec-specific Transmission impairment factors from recommendation G.113. Namely, the R-factors (for Listening and Conversation Quality) are calculated as following:

```
R(LQ) = R0 - Is - Ie-eff + A,  and R(LQ) = R(CQ) - Id
where:
  R0 is Basic Signal to Noise Ratio | narrowband: R0-Is = 93.25
  Is is Simultaneous Impairments    | wideband:   R0-Is = 120
   A is Advantage Factor = 0 constant

Id = Idte + Idle + Idd,  where: Idte is Talker Echo = 0 constant
                                Idle is Listener Echo = 0.15 constant

Idd = If Ta <= 100 ms, Idd = 0     {           1/6               1/6   }
                                   { [       6 ]      [         6 ]     }
      If Ta  > 100 ms, Idd =  25 x { [ 1 + [Y] ]  - 3x[ 1 + [Y/3] ] + 2 }

             log(Ta/100)
  where: Y = -----------        (Ta is estimates as half round-trip time)
                log2
                            Ppl
Ie-eff = Ie + (95 - Ie) x -------
                         Ppl + Bpl
where:
  Ie is Codec-based Equipment Impairment Factor defined in ITU-T G.113
  Ppl is Packet Loss %
  Bpl is Codec-based Packet Loss Robustness Factor defined in ITU-T G.113
```

After that, MOS is calculated from R(LQ) as following:

```
Rx = codec_is_wideband ? R/1.29 : R;
MOS = If Rx < 0;        MOS = 1             [7Rx(Rx - 60)(100 - Rx)]
      If 0 <= Rx < 100; MOS = 1 + 0.035Rx + ------------------------
      If Rx >= 100;     MOS = 4.5                     10^6
```

Codec-based impairment and robustness factors are listed in the following table, along with maximum possible R(LQ) and MOS values, which can be achieved in "perfect" conditions, i.e. with no packet loss, low jitter (completely covered by jitter buffer) and small round-trip time (below 100 ms):

Codec	Ie	Bpl	max_R	max_MOS
G.711 (pcmu or pcma)	0	10.0	93	4.4
G.722	5	7.1	88	4.3
G.729	10	19.0	83	4.1

Codec	Ie	Bpl	max_R	max_MOS
Opus	16	17.0	93	4.4
iLBC	10	28.0	83	4.1
iSAC (wideband)	0	10.0	120	4.4

The third-party VoIP Quality and Bandwidth Calculator may be used for calculating expected MOS for different packet loss rates.

Reporting Operational Data

This page describes SIP Endpoint SDK capabilities to report problems with audio stream and other operational data to the application code, intended to provide real-time indication to the user of something not working as expected because of environmental problems (audio device malfunction, network data path broken etc).

Detection and reporting of missing audio

The purpose of this feature is to detect a case when no audio data comes in either direction, caused by:

- microphone device malfunction, or microphone being inadvertently muted by user.
- audio stream coming from remote side consists of complete silence, because of data path broken somewhere.

Voice Activity Detection is performed at the voice engine level, using *Real-time Access to Audio Stream* feature internally, with the results of the analysis attributed to currently active session. Detection results for each 10 msec audio frame are processed as following (separately in both directions):

- 15 frames (150 msec of audio) are aggregated into one data point for histogram, printed to the log at each 10th voe:checkpoint (about each 10 sec)
- total number of "active" frames and timestamp of last such frame are saved in audio statistics structure

Application access to the raw detection results (or histogram data) is not planned at this moment.

SIP Endpoint Core (C++) API

Existing audio statistics structure is updated with new fields:

```
typedef struct gs_call_statistics {
...
  // Voice Activity Detection info, added in 9.0.020:
  //
  int got_vad_info; // 1 if got outgoing VAD, 2 if got incoming, 3 if got both
  int active_10ms_out;   // total number of frames with mic activity detected
  int active_10ms_in;    // - incoming frames with activity detected (speaker)
  pgtime_t last_vad_out; // timestamp of last "out" frame with voice activity
  pgtime_t last_vad_in;  // timestamp of last "in" frame with voice activity
}
gs_call_statistics;
```

Since that data can be queried at any moment during the call and is available after call release, that should provide enough information for:

- sending telemetry event when "one-way audio" or "no audio" call is detected (no active frames for the duration of the call)
- triggering application logic to warn user about:
  - missing outgoing audio (mic was muted or not working properly) or
  - silence in incoming audio (to indicate the problem is not with local headset volume, but on remote side)

Note: in case when no activity was detected for particular direction (i.e., `active_10ms_in/out` == 0), the corresponding timestamp refers to the moment when audio session was started.

SDK for OS X (objective-C) API

The existing interface in **Endpoint/Src/Headers/GSStatistics.h** header is updated as following:

```
@interface GSStatistics : NSObject {
...
  int gotVADinfo;            // 1 if got outgoing VAD, 2 if got incoming, 3 if got both
  int active10msOut;         // total number of frames with mic activity detected
  int active10msIn;          // - incoming frames with activity detected (speaker)
  struct timeval lastVADout; // timestamp of last "out" frame with voice activity
  struct timeval lastVADin;  // timestamp of last "in" frame with voice activity
}
```

In addition, an operational event `sip_media_silence` may be sent at the end of the call, to indicate that no voice activity was detected in the incoming RTP stream for the entire duration of the call, see the description below on this page. No such notification is planned for detecting microphone silence, as this was considered too dependent on business logic (for example, in Service Observing cases, a supervisor is expected to only listen to the call, with microphone muted on some level).

Histogram printout in the log

In order to not flood the log with with too much data, it makes sense to aggregate the results into bigger buckets. 150 msec (15 frames) was considered a good compromise, providing enough granularity to see what happened, but not too much for affecting the log size. On low-level API, GSeptVoE class has a method to return audio stream analysis as a string (no longer than 64 chars), describing the detection results of last 9.6 seconds of incoming / outgoing audio, each position corresponding to 150 msec interval encoded as:

- `_` (underscore) for total silence (all zeroes in frames data = no RTP or audio muted)
- `-` for near silence, total signal energy for all packets E < 0.001
- `+` for mostly silence (no more than a couple of frames with sound)
- `o` for 3 to 10 frames with E ≥ 0.001
- `0` for at least 11 frames with high energy

That info is printed on each 10th checkpoint (about each 10 seconds) on INFO level, for example:

```
11:16:46.130|77831|dbg voe:checkpoint,sent[2]=48/7680(64.0kbps)
11:16:47.091|77831|-i- voe:checkpoint,sent[2]=48/7680(64.0kbps)
   in: ------oo+-+-o-+--oo-o+-oo0000o00000-+++oo---o----------------+o0
```

```
  out: +++00000000000++----+-+-------------+oo0000000000o00o0o---++----
...
11:16:55.745|77831|dbg voe:checkpoint,sent[2]=48/7680(64.0kbps)
11:16:56.705|77831|-i- voe:checkpoint,sent[2]=48/7680(64.0kbps)
   in: o00000000o0000000o00o0-+-------o000+00000o00000oo--o-----0--o---
  out: -----------------------------------00o---------+-+0000oo00000000
 ...
```

A nice bonus is that one can see in the log how the conversation was going on. That should not be a privacy concern, as only the voice energy level is measured, no other details can be possibly deduced from this data.

Provisioning

No new or updated options are provided for this feature. Note that the existing option, `policy.session.vad_level` is **not** related to this feature, as it is only used for Discontinuous Transmission (DTX) feature, applicable when `dtx_mode=1` .

Reporting SDK operational data

This feature utilizes the same mechanism as used for Telemetry Service in Genesys Engage cloud, so the operational data is essentially the same, with only a couple of modifications done to existing implementation:

- in addition to passing pre-formatted message string to the application, API provides a method to access variable parts of the trace separately, to simplify handling of this info on the application side
- these callbacks are added to SDK for OS X (objective-C) API

No significant changes in telemetry data is done as part of this feature, but all further updates will be automatically applied.

SIP Endpoint Core (C++) API

A new method to enable application-side telemetry and a callback for getting the data were added as following:

```
class GsEndpointCore
  {
  ...
    // Callback for getting telemetry data in the application code, with value of
    // trace arguments added to context XKVList with integer keys for easy access
    // (method enable_app_telemetry MUST be called before Setup to take effect):
    //
    gs_status enable_app_telemetry(int trace_level_from_0_to_4 = 3);
    virtual void  on_app_telemetry_trace(int level, XKVList *context_n_args,
                                         int msgId, CGSString message) { }
```

Parameter `message` contains a complete preformatted trace message, but the original parameters' values are also duplicated in the provided XKVList with integer keys (starting from index 1), so the application code does not need to parse that message to access specific parameter, for example:

```
  19:27:00.125| 775|-i- (tm)trace(i:4001): Device(mic:1002) selected: Plantronics C610
```

```
19:27:00.125| 775|-i-   category="sip_dev"
19:27:00.125| 775|-i-   1="mic"
19:27:00.125| 775|-i-   2=1002
19:27:00.125| 775|-i-   3="Plantronics C610"
```

Refer to the List of traces, events and metrics section below for the description of arguments for each trace msgId.

SDK for OS X (objective-C) API

A new method has been added to GSEndpoint protocol to enable application telemetry:

```
@protocol GSEndpoint <NSObject>
 /**
  Enable appTelemetryNotification in GSEndpointNotificationDelegate for getting
  SIP Endpoint telemetry data in the application code, with given trace level;
  must be called on application startup, before calling 'configure' method
  */

 - (GSStatus) enableAppTelemetry:(int) trace_level_from_0_to_4;
```

Data will be delivered via the following notification method:

```
@protocol GSEndpointNotificationDelegate <NSObject>
/**
 Called on receiving telemetry data from SIP Endpoint Core
 @param level is trace level, currently from 1 (high) to 3 (info)
 @param msgId is message ID, one of values defined in gs_telemetry.h header
 @param msg formatted message text
 @param context_n_args includes trace context attributes and variable arguments
*/
- (void) appTelemetryNotification:(int) level
                               id:(int) msgId
                     message:(NSString*) msg
                     context:(XKVList *) context_n_args;
```

Parameter `message` contains complete pre-formatted trace message, but the original argument values are also duplicated in the provided XKVList with integer keys, same as for C++ API. Since Apple's application is currently written in Swift, to spare development time on (and CPU cycles) on multiple conversions of data structures, the type of last parameter is internal pure-C object (which can be used in objective-C code directly), with data access function defined in `gs_xkvlist.h` header.

Accessing data in XKVList

Getting **trace attributes** - stored in XKVList with string keys, - may be done with the following functions (all "GetString" functions miss 'const' type qualifier for historical reasons, the returned pointer must not be used for modifying the value):

```
char  *XKVListGetStringValue(XKVList *kvl, const char *key, XKVResult *res);
double XKVListGetFloatValue (XKVList *kvl, const char *key, XKVResult *res);
```

Last argument may be used for checking operation result, but it is rarely used for string and integer values, since they both have a natural indication of missing value, NULL for strings and `-1` for integers (XKVList cannot hold NULL strings and negative integers anyway). For example, the code for getting SIP Call-ID string from trace context:

```
   const char *call_id = XKVListGetStringValue(context_n_args, "call_id", NULL);
   if (call_id != NULL) ...
```

Accessing **trace arguments** - stored with integer keys (starting from 1), - may be done with the following functions (with letter 'i' added in the function name, since C does not allow function overloading):

```
char *XKiVListGetStringValue(XKVList *kvl, int id, XKVResult *res);
int   XKiVListGetIntValue   (XKVList *kvl, int id, XKVResult *res);
```

For example, getting IP address and port of timed-out connection (for GsTMT_HOST_BLACKLISTED trace, msgId == 1502):

```
const char *IPaddr = XKiVListGetStringValue(context_n_args, 1, NULL);
int port = XKiVListGetIntValue   (context_n_args, 2, NULL);
```

Refer to next section for the list of arguments for each trace message type. The full content of given XKVList may be also traversed with **scan loop**, using these functions:

```
 XKVResult XKVListInitScanLoop(XKVList *kvl);
  XKVPair  *XKVListNextPair    (XKVList *kvl);
```

and the following functions to access the XKVPair object:

```
   XKVType XKVListType (XKVPair *kvp); /* value type */
   XKVType XKVListKType(XKVPair *kvp); /*   key type */
   char *XKVListKey(XKVPair *kvp);
   int  XKiVListKey(XKVPair *kvp);
   char *XKVListStringValue(XKVPair *kvp);
   int   XKVListIntValue   (XKVPair *kvp);
```

For example, printing all data received in appTelemetryNotification to the log:

```
 - (void) appTelemetryNotification:(int) level
                               id:(int) msgId
                     message:(NSString*) msg
                     context:(XKVList *) context_n_args
 {
   [logger logInfoMessageWithFormat:@"(tm)trace(%d:%d) %@",level,msgId,msg];
   XKVListInitScanLoop(context_n_args);
   XKVPair *xp;
   while (( xp = XKVListNextPair(context_n_args) )) {
     if (XKVListKType(xp) == XKVTypeString) { // context attribute
       switch (XKVListType(xp)) {
       case XKVTypeString:
         [logger logInfoMessageWithFormat:@"  %s=\"%s\"",
                 XKVListKey(xp), XKVListStringValue(xp)];
         break;
       case XKVTypeFloat:
         [logger logInfoMessageWithFormat: @"  %s=%.1f",
                 XKVListKey(xp), XKVListFloatValue(xp)];
         break;
     } }
     else if (XKVListKType(xp) == XKVTypeInt) { // trace argument
       switch (XKVListType(xp)) {
       case XKVTypeString:
         [logger logInfoMessageWithFormat: @"  %d=\"%s\"",
                 XKiVListKey(xp), XKVListStringValue(xp)];
         break;
       case XKVTypeInt:
         [logger logInfoMessageWithFormat:  @"  %d=%d",
                 XKiVListKey(xp), XKVListIntValue(xp)];
         break;
     } }
 } }
```

List of traces, events and metrics provided

Currently SIP Endpoint Core provides traces as listed in the table below, where **lvl** column refer to trace level as following (0 = "critical" and 4 = "debug" levels are currently not used and reserved for future extensions):

- `H` (1) "high" - important errors and connectivity messages, in normal operation that should be just a handful of entries per user session
- `M` (2) "medium" - warnings and medium-priority messages, should be just a couple of them per call
- `i` (3) "info" - the bulk of normal trace, including session events, SIP messages etc

Trace GsTMT_TM_EVENT is sent with "Medium" level for "sip_session_established/_completed" events, and "High" level for all other events.

msgID	lvl	message format and comments	arg1	arg2	arg3+
GsTMT_ENDPOINT_STARTED 1001	`H`	`Endpoint Core %s started with config:%s`	(str) version or Endpoint Core	(str) config in simplified format	
GsTMT_CONNECTION_OK 1200	`H`	`%s connected to %s` message for successful SIP registration	(str) user DN	(str) server URL	
GsTMT_CONNECTION_FAIL 1488	`H`	`%s failed to connect to %s (rc:%d %s)` SIP registration failed with given error code / msg	(str) user DN	(str) server URL	(int) error code (str) err message
GsTMT_DNS_RESOLVED 1500	`H`	`DNS: %s resolved to %s (port:%d)`	(str) hostname or FQDN resolved	(str) IP address	(int) port used in connection that triggered lookup
GsTMT_HOST_BLACKLISTED 1502	`H`	`DNS: %s:%d blacklisted (request`	(str) IP address	(int) port	

msgID	lvl	message format and comments	arg1	arg2	arg3+
		timeout)			
GsTMT_TM_EVENT 2010	`*`	event name, see list below			
GsTMT_TM_METRIC 2020	`M`	metric name, see below (metric value is provided in attribute named "value__NUM")			
GsTMT_CALL_STARTED 3001	`M`	`SIP call started (remote:%s)`	(str) remote DN		
GsTMT_SESSION_EVENT 3003	`i`	`session event %s`	(str) event name		
GsTMT_SIP_MSG_SENT 3035	`i`	`SIP message %s sent` `%s` full SIP message text may be not available (e.g. outgoing CANCEL or auto-sent ACK)	(str) SIP method or response code	(str) SIP message	
GsTMT_SIP_MSG_RECEIVED 3036	`i`	`SIP message received from %s` `%s`	(str) remote IPaddr:port	(str) SIP message	
GsTMT_CALL_COMPLETED 3200	`M`	`SIP call completed%s`	(str) call info and statistics		
GsTMT_CALL_FAILED 3400	`M`	`SIP call failed (rc:%d)`	(int) response code for failure		
~~GsTMT_CALL_NO_MEDIA~~3408		removed as duplicate of "sip_media_inactivity" event (now sent as GsTMT_TM_EVENT trace)			

msgID	lvl	message format and comments	arg1	arg2	arg3+
GsTMT_MIC_NO_SIGNAL 3500	`H`	`No signal from mic during the call(talk=%.1f)`	(float) talk time of call (in secs)		
GsTMT_VQ_ALARM_RAISED 3504	`H`	`Voice Quality alarm (%s)`	(str) MOS value as "`MOS=%.1f`"		
GsTMT_DEVICE_SELECTED 4001	`i`	`Device(%s:%d) selected: %s`	(str) device type: mic/out/cap/ring	(int) device ID	(str) device name
GsTMT_AUDIO_STARTED 4004	`i`	`audio-started(err:%d,dt:%dms)`	(int) error mask, zero for success	(int) start delay in milliseconds	
GsTMT_DEVICE_FAILED 4400	`H`	`Audio %s failed`	(str) failed direction recording/playback		
GsTMT_SEC_LOAD_ERROR 2002	`H`	unable to load Genesys Security Pack	these trace messages originated from Genesys MFWK library, no separate arguments available		
GsTMT_SEC_CONN_ERROR 8102	`H`	Secure connection error			
GsTMT_SEC_CONNECTED 8103	`i`	Secure connection is established			
GsTMT_SIP_REJECTED 53051	`H`	`SIPdialog[%d] %s -- rejected %d%s`	trace messages from SIP stack, no separate arguments available		
GsTMT_SIP_NOT_IMPLEMENTED 53056	`H`	`Method not implemented, rejected`			
GsTMT_SIP_REFER_ERROR 53057	`H`	`REFER error -- %s`			

The following attributes are provided as trace context, when applicable (all attributes except the last one have string value):

- "`ThisDN`" = 'user' parameter of the related connection
- "`call_id`" = SIP Call-Id of the related session

- "`call_uuid`" = value of X-Genesys-CallUUID SIP header for related session
- "`region`" - only provided for multi-region deployment, when 'server' connection parameter is specified with SRV-resolved FQDN, the value is equal to A-record SRV target used for connection
- "`category`" = trace category, one of:
    - "`sip_acct`" - account (connectivity) and general traces
    - "`sip_call`" - call (session) related traces
    - "`sip_dev`" - audio/video device related traces
- "`value__NUM`" = value of reported metric (XKVTypeFloat, only provided for GsTMT_TM_METRIC trace)

List of metrics currently provided by SIP Endpoint SDK:

- "`sip_offline_time`" - this metric is generated periodically (currently each 5 minutes, but that may be subject to change) with value of "offline" time in seconds = the duration since last report when SIP registration was not active because of inability to communicate with SIP proxy
- "`sip_call_mos`" - generated at the end of the call (same as the next two metrics), value is estimated MOS between 1.0 and 5.0
- "`sip_call_ring_time`" - ringing time for the call in seconds
- "`sip_call_dead_time`" - "dead" time for the call = the duration for which call was considered "established" on signaling level, but no RTP was received

The following operational events are currently reported via GsTMT_TM_EVENT trace:

- "`sip_connected`" - successfully connected and registered to SIP proxy
- "`sip_disconnected`" - SIP connection failed or closed on shutdown
- "`sip_switchover`" - only provided for multi-region deployment, sent when "`region`" attribute changes
- "`sip_session_established`" - new call (session) is established
- "`sip_session_completed`" - call (session) has been released
- "`sip_request_error`" - SIP error response received
- "`sip_request_timeout`" - SIP request transaction timeout
- "`sip_invite_rejected`" - incoming SIP call rejected for whatever reason (busy, no headset or codec mismatch)
- "`sip_media_inactivity`" - call has been dropped because of RTP inactivity timeout
- "`sip_media_silence`" - this event is sent at the end of the call, to indicate that no voice activity was detected in the incoming RTP stream for the entire duration of the call ("one-way audio" condition)

Configuring secure connections (TLS) for SIP

The TLS support in SIP Endpoint SDK and Genesys Softphone on macOS is based on the Genesys Security Pack implementation, which relies on the OpenSSL library, a de-facto industry standard implementation on UNIX platforms. Currently, the macOS Keychain system is not supported. Instead, users should provide the client-side certificate and trusted CA list as files, using one of the file formats supported by the OpenSSL project.

To secure the connection to the SIP Server, users should set the **protocol** parameter to TLS in the corresponding Connectivity element of the **Basic** container in the SIP Endpoint SDK configuration file, whether the connection is direct or via SIP Proxy or SBC.

```
<Connectivity user="{dn}" server="{server:port}" protocol="TLS"/>
```

For Mutual TLS, you should also specify the **certificate** and **certificate-key** options, which refer to the public and private parts of the client-side certificate.

If these options are left empty, outgoing TLS connections will be used, but incoming TLS connections will not be possible. However, most deployments do not encounter this problem because Genesys SIP Server, SIP Proxy, and supported SBCs reuse the client-originated TLS connection by default without opening another TLS connection for delivering incoming SIP messages. In either case, the **trusted-ca** option is mandatory and should include the public key of the Certificate Authority (CA) used to sign the server-side certificate.

Important

The TLS configuration settings for securing the SIP connection can be found in the **system.security** section.

certificate

Valid Values: String

Full path pointing to the certificate file, which is used as a client-side certificate for outgoing TLS connections in case of Mutual TLS, and server-side certificate for incoming TLS connections in case when SBC is configured to not reuse the client TLS connection. For example, `/Users/jdoe/gcti/certificate/hostname_cer.pem`.

This option replaces the `cert_file` option from previous versions. For backwards compatibility, the SDK accepts `certificate` or `cert_file` and the former takes priority.

certificate-key

Valid Values: String

Full path to the endpoint Private Key file, corresponding to the Public Key in the certificate specified by **certificate** option, or, if the Private Key is stored with the certificate, the full path to the certificate .pem file. For example, `/Users/jdoe/gcti/certificate/hostname_priv_key.pem`

This option replaces the `priv_key_file` option from previous versions. For backwards compatibility, the SDK accepts both `certificate-key` and `priv_key_file` option names and the former takes priority.

trusted-ca

Valid Values: String

Full path to the Certificate Authority's (CA) list file. For example, `/users/jdoe/gcti/certificate/ca/ca_cert.pem`

This option replaces the `ca_list_file` option from previous versions. For backwards compatibility, the SDK accepts `trusted-ca` or `ca_list_file` option names and the former takes priority.

sec-protocol

Valid Values: SSLv23, TLSv12, TLSv13 or an empty string

Default Value: an empty string

Specifies the protocol used by the component to set up secure connections:

- SSLv23 - The highest TLS protocol version supported by both sides of communication, from TLS 1.1 and up (remains for backward compatibility, not recommended for new deployments).
- empty string - the default Security Pack settings (currently the highest TLS version supported by both sides from 1.2 upwards).
- TLSv12 - TLS version 1.2.
- TLSv13 - TLS version 1.3.

cipher-list

Specifies the defined list of ciphers for TLSv1.2 and earlier protocols. The cipher list must be in a valid format for OpenSSL library. See the cipher-list setting defined in the Security Deployment Guide and OpenSSL documentation.

ciphersuites

Valid Values: The colon-separated list of TLSv1.3 ciphersuite names, as defined in RFC 8446, in preference order. The list may include one or more of the following:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

Default Value: empty string, which is equivalent to
`TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256`

Specifies the defined list of ciphersuites to be used for TLSv1.3, if that TLS version is supported by both side of the connection (and negotiated during handshake). This option supplements `cipher-list` option (which is still applicable for TLSv1.2 and below).

tls-crl

Specifies the full path to the file that contains one or more certificates defining the Certificate Revocation List (CRL). See the tls-crl setting defined in the Security Deployment Guide and OpenSSL documentation for more information.

tls-target-name-check

Valid Values: `no`, `host`

Default Value: no

Specifies if the Common Name in the subject field and/or the Subject Alternate Names of the server’s certificate will be compared to the target host name (option value `host`). If they are not identical, the connection fails. If the option is set to no, a comparison is not made, and the connection is allowed (provided the server's certificate is valid and signed by trusted CA, as disabling target name check does not affect other certificate verification steps).

Important

The default value for this option is 'no' only to ensure backward compatibility with previous releases. Genesys recommends to always set the value as '`host`' in production environments for security reasons to avoid any man-in-the-middle attack attempts.

Important

If encryption is enabled in SIP mode, the user workstation may connect to the CRL systems of the Certificate Authorities that issued the SSL certificates for the SIP User Agents (SIP UAs).

For additional information, refer to:

- Genesys Secure Connection (TLS) guide
- OpenSSL project
- certificate formats (on OpenSSL documentation)