SIP Endpoint SDK Developer's Guide

Reporting Operational Data

12/13/2025

Contents

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: o00000000o00000000o00o0-+-------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)