SIP Endpoint SDK Developer's Guide

Support real-time access to audio stream

12/19/2025

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());
}
```