SIP Endpoint SDK Developer's Guide

Separate Ringer Volume Control

4/15/2025

Contents

Separate Ringer Volume Control

> **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;
}
```