SIP Endpoint SDK Developer's Guide

Configuring SIP Endpoint SDK for .NET

12/12/2025

Contents

Configuring SIP Endpoint SDK for .NET

> **Warning**
> **This documentation is outdated and is included for historical purposes only.**

Basic Configuration Settings

You may want to set certain basic SIP Endpoint configuration options in your application. The following sections contain information that may be helpful in doing that.

Most configuration settings described on this page can be found and changed in the SipEndpoint.config file that is included with your SIP Endpoint SDK installation. For more information about this file, see Default Config Settings.

Timeout Issue

Note that the `reg_timeout` setting is valid only if it contains a numeric value. If the supplied value is a non-numeric value or is empty or null, it will be interpreted as `0` and the endpoint will operate in standalone mode.

IPv6

As of release 8.1.2, SIP Endpoint SDK for .NET supports TCP/IP version 6 (IPv6) on Windows Vista and higher.

The local address must be set to IPv6:

- By means of an explicit address specification (as a hostname resolving to IPv6 only, or by the use of a literal IP address)
- By setting the `ip-versions` option to favor IPv6
- Or by only having an IPv6 interface available on the host

The SIP proxy address must be either a hostname that primarily resolves to IPv6 or a literal IPv6 address. Note that the preferences in the `ip-versions` option do not currently affect external address resolution.

Note: IPv6 must be configured on both ends of a session — mixed IPv4 and IPv6 environments are not supported.

This feature is resolved at Endpoint creation time by applying the following configuration to the SIP

Provider.

Settings:

```
<setting name="public_address" value="Address"/>
<setting name="ip_versions" value="ipv4,ipv6"/>
```

QoS Bits

You can mark RTP packets by setting the QoS (TOS) bits so that network routers can prioritize them. This can make a big difference in the quality of your voice transmissions. When the SIP Endpoint is initializing, it gets information on QoS (TOS) bit settings from the SipEndpoint.config file. This information cannot be changed while the endpoint is running. The `system:qos:audio` setting allows you to specify a string value that properly marks your RTP packets, and the type of QoS supported for audio. For example, you could specify the type of QOS with a value such as "tos 22". If QOS is not supported for audio, leave this setting empty.

Note: QoS is not supported for Windows Vista, Windows 7, or higher.

Diagnostics

In order to diagnose production issues, it is important to have access to diagnostic information that relates to your SIP endpoint and its environment. Because of this, the SIP Endpoint SDK provides logging with 5 possible logging levels.

Valid values for the SIP Endpoint SDK logging levels are:

```
Fatal = 0
Error = 1
Warning = 2
Info = 3
Debug = 4

<domain name="system">

</domain>
```

Headset Connectivity Notification

The SIP Endpoint SDK only supports headset monitoring when the device is explicitly defined in the `SipEndpoint.config` file, as shown below:

```
<domain name="policy">
...

</domain>
```

In this case, headset connectivity is fully supported and the application can receive events indicating that the device has been plugged or unplugged.

It is also possible to use your application to select audio devices automatically, by using the SIP Endpoint SDK API to look for IN and OUT devices. Using the `GetAllSystemAudioInDevices()` and `GetAllSystemAudioutDevices()` methods allows you to check for attached devices, which can then be used to update the following settings before initializing the SIP Endpoint SDK.

```
<domain name="policy">
...

</domain>
```

However, the plugged or unplugged states are not monitored for the selected device in this case. If no devices are defined in the `SipEndpoint.config` file, then the SIP Endpoint SDK uses a default procedure to determine what device is attached, and will not monitor for plugged or unplugged states.

SIP Messaging and Media Encryption

SIP Endpoint SDK has settings that allow you to encrypt SIP messaging and the media channel. The SIP messaging can be encrypted using TLS, while the media channel can be encrypted using SRTP. The following diagram shows the SIP Endpoint SDK architecture with these features enabled.

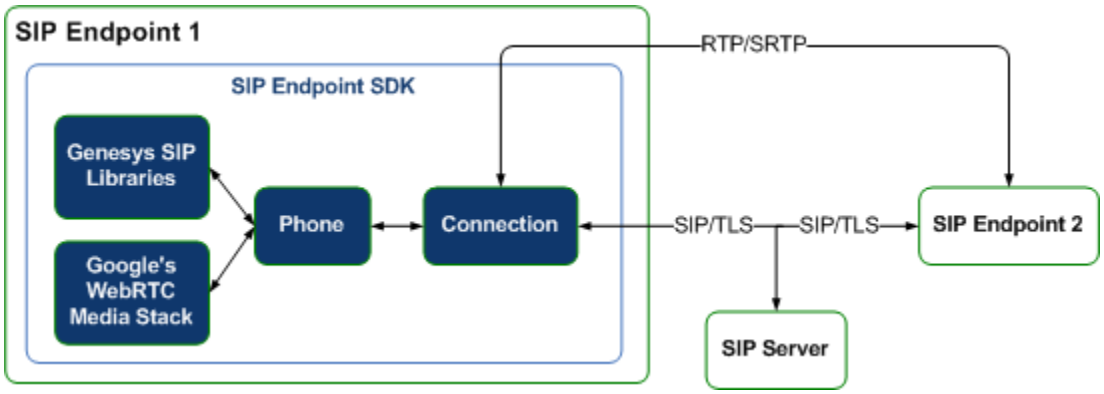

Figure 1: SIP Endpoint SDK Architecture with TLS and SRTP Enabled

To enable TLS support, set the protocol to `tls`, as shown here. The outgoing SIP messaging will be encrypted:

```
<Container name ="Basic">
  <Connectivity user ="DN" server="SipServerHost:Port" protocol="tls"/>
</Container>

<domain name="system">
. . .

</domain>
```

Use the `use_srtp` setting to control media channel encoding. Valid values are shown in the above code sample.

Digest Authentication Support

SIP Endpoint SDK supports the RFC2617-style digest authentication that is currently used by SIP Server. This authentication is triggered by using the following DN configuration options: `password` and `authenticate-requests`. If these options are configured for a specific DN, then SIP Server enforces the authentication mechanism for that DN. During registration, the SIP Endpoint receives a `401 Unauthorized` SIP challenge message request from SIP Server. The SIP Endpoint should then provide the encrypted password, along with the `Digest username`, in the next `REGISTER` message.

At this point, SIP Server compares the received password to the password associated with the DN object defined in Config Server. If the passwords match, then the SIP Endpoint can be registered and may proceed to access the call functionality provided by SIP Server. SIP Endpoint SDK provides the following method to set the password for a particular connection ID. This method has been added to the `IExtendedService` interface in the `Genesyslab.Sip.Endpoint.Provider.Genesys` namespace:

```
void SetPassword(int connectionId, String^ password);
```

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.

Note: The publish message is sent to the specified collector address only if the `vq_report_collector` parameter is configured with the `user@server:port` format. For example, `collector@172.24.133.136:5160`.

Settings:

```
<domain name="policy">
<section name="endpoint">
...
<!-- (int) publish VQ report: 0=never, 1=end-of-call -->
<setting name="vq_report_publish" value="0"/>
<!-- (str) collector address (if NULL/empty => publish to proxy) -->
<setting name="vq_report_collector" value="vq_report_collector_URI"/>
</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);
```

Audio Layer Selection

SIP Endpoint SDK 8.1.2 now allows you to select an audio layer for WebRTC using the Windows environment variable `GCTI_AUDIO_LAYER`. If this variable is set to 1, SIP Endpoint SDK will use the Windows Wave audio layer. Otherwise, it will use the Windows Core audio layer. You can set this variable at Computer -> Properties -> Advanced System Settings -> Environment Variables.

Other Items for Review

The following items have been placed on this page in order to expedite their review. They will be moved to an appropriate page prior to publication.

SIP::INFO Message Exchange

SIP Endpoint SDK can create and transmit In-Dialog `SIP:INFO` messages using updated `Content-`

Type and Content headers. It can also receive messages with customer-provided information in the Content-Type and Content headers.

The following method has been added to the ICallControl interface in order to create and transmit an In-Dialog SIP:INFO message::

```
void SendSipInfo(int sessionId, String^ contentType, String^ content);
```

The following event has been added to Session Manager:

```
public event EventHandler<SessionEventArgs> SipInfoReceived;
```

The SessionEvent properties now also have these key value pairs:

```
Properties["ContentType"].ToString();
Properties["Content"].ToString();
```

RTP Statistics for .NET

SIP Endpoint SDK for .NET now allows you to monitor RTP video and audio statistics.

Methods have been added to the ICallControl interface so you can get audio and video statistics, both during a session and when the session has been disconnected:

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

The properties structure of these statistics is the same for both audio and video, as shown here:

```
Dictionary<String^,Object^>^ properties;

// positive if next 5 fields are valid (local stat available)
properties->Add("Got Local Stat ". . .);
// fraction of lost packets (i.e. not locally received)
properties->Add("Local Frac Lost". . .);
// interarrival jitter
properties->Add("Local Jitter   ". . .);
// local octet count (number of bytes sent)
properties->Add("Local Oct Count". . .);
// local packet count (number of RTP packets sent from our side)
properties->Add("Local Pkt Count". . .);
// total lost packaged (calculated per RFC 3550)
properties->Add("Local TotalLost". . .);

// true if next 3 fields are valid (got Receiver Report from remote side)
properties->Add("Got Remote RR  ". . .);
// fraction of lost packets (i.e. not received by remote end)
properties->Add("Remote FracLost". . .);
// interarrival jitter
properties->Add("Remote Jitter  ". . .);
// total lost packaged (calculated per RFC 3550)
properties->Add("Remote TotaLost". . .);

// true if next 2 fields are valid (got Sender Report from remote side)
properties->Add("Got Remote SR  ". . .);
// remote octet count
properties->Add("Remote OctCount". . .);
// remote packet count (number of packets sent by remote side)
properties->Add("Remote PktCount". . .);
```

```
// Round-trip time in milliseconds.
properties->Add("Round Trip Time". . .);
```