GIPS VoiceEngine

API Guide

Headquarters
Global IP Solutions, Inc.
642 Harrison Street, Second Floor
San Francisco, CA 94107
USA

Phone: +1 415 397 2555
Fax: +1 415 397 2577

For additional contact information, please visit the GIPS URL: http://www.gipscorp.com

For information on how to contact Customer Support, please visit the following URL:
https://www.gipssolutions.com/customer/login.asp

Contents

1 About This Guide

This API guide describes the interface of the GIPS VoiceEngine (VE), which is delivered as a C++ static or dynamic library. The VoiceEngine has the following main functionalities:

1. Soundcard handling
2. Speech processing
3. RTP communication

The API Guide gives the information required to integrate the VoiceEngine into a complete VoIP solution. It describes the VoiceEngine's capabilities and how to control its many options.

NOTE: The VoiceEngine does *not* handle call setup.

This chapter gives an overview of the contents in this document, the product version number, writing conventions, and a change history for this guide.

Product Version

This GIPS VoiceEngine API description corresponds to GIPS VoiceEngine product version 3.5.0.0.

In This Guide

This API guide gives a thorough description of the GIPS VoiceEngine API:

- Chapter 2, Sub-APIs, describes the new API design and lists supported sub-APIs.
- Chapter 3, API Overview, summarizes how to allocate resources, acquire and use sub-APIs, and finally how to release resources in a correct way.
- Chapter 4, API Reference is a reference guide and covers all VoiceEngine API functions. Code examples are also given in this chapter.
- Chapter 5, Example code, gives examples of typical usage.
- Chapter 6, Customizing Codec Settings describes the change of codec settings in depth and gives allowed settings.
- Chapter 7, Error Handling describes the recommended error handling.
- Appendix A, Error Codes, describes all possible error codes.

- Appendix B, Runtime Error Codes gives runtime error codes.

Document Change History

The following Document Change History table records the technical changes to this document, giving the API version number, revision date, and a summary of the change.

Version	Date	Change Summary
3.0.0	2008-04-11	Adapted the VoiceEngine API guide to the new sub-API design method, which was introduced in version 3.0.
3.0.1	2008-08-15	Updated with API changes and additions done in the 3.0.1 release: GIPSVE_GetInputMute, GIPSVE_GetSpeechInput/OutputLevelFullRange, GIPSVE_Set/GetSendCodec, GIPSVE_Set/GetG729AnnexBStatus, GIPSVE_VoiceActivityIndicator, GIPSVE_GetRecording/PlayoutDeviceName, GIPSVE_StartRecordingPlayout/Microphone/Call.
3.0.2	2008-09-21	Added Windows CE/Mobile platform support notes.
3.0.3	2008-10-28	Added Symbian 9.x/S60 3rd Ed. platform support notes.
3.1.0	2008-12-15	Updated with API changes and additions done in the 3.1.0 release: Removed GIPSVE_SetNetEQPlayoutFaxMode. Added GIPSVE_Set/GetNetEQPlayoutMode. Added GIPSVE_SetISACInitTargetRate and GIPSVE_SetISACMaxPayloadSize. Added GIPSVE_SetDeadOrAliveObserver and GIPSVE_SetPeriodicDeadOrAliveStatus to the GIPSVEVQE sub-API. Removed support for IRS filters in GIPSVESet/GetSendCodec. Added GIPSVE_Set/GetMetricsStatus, GIPSVE_GetSpeechMetrics, GIPSVE_GetNoiseMetrics, and GIPSVE_GetEchoMetrics to the GIPSVEVQE sub-API. Updated GIPSVE_SetAGCStatus with information about new run-time error code called VE_SATURATION_WARNING. Removed GIPSVE_Set/GetTraceFileName. Added GIPSVE_SetDebugTraceFileName and GIPSVE_SetTraceFilter. Added GIPSVECallReport sub-API. Updated GIPS_transport class (return values). Updated information about SRTP.
3.1.1.0	2009-02-04	Updated with API changes and additions done in the 3.1.1.0 release: Added GIPSVE_GetRTCPStatus. Added GIPSVE_Set/GetRTPKeepaliveStatus. Added ResetSoundDevice. Added iPhone platform support notes. Added information about GIPS VoiceEngine PC Lite. Improved documentation about CallbackOnTrace.
3.2.0.0	2009-02-17	Updated with API changes and additions done in the 3.2.0.0 release: Improved documentation for GIPSVE_GetCPULoad. Updated Codec Settings section with details about G.729.1. Added GIPSVE_Set/GetSamplingRate to the GIPSVEHardware sub-API. Added Linux support for GIPSVE_Set/GetSendTOS. Updated Appendix B Runtime Error Codes. Added information about VoiceEngine Lite.
3.2.1.0	2009-03-20	Updated information on PulseAudio in GIPSVE_Init.
3.2.1.1	2009-04-23	No changes.
3.3.0.0	2009-07-06	Added rtcpPort argument to GIPSVE_GetSourceInfo. Added GIPSVECodec::GIPSVE_GetRecCodec. Updated information about getting and setting sound devices (Linux/ALSA and Mac). Added information for the GIPSVE_SetConferenceStatus API. Updated information about GIPSVE_SetRecPayloadType. Modified GIPSVERTP_RTCP::GIPSVE_GetRemoteSSRC. Added GIPSVERTP_RTCP::GIPSVE_GetRemoteCSRCs. Added GIPSVERTP_RTCP::GIPSVE_GetRemoteEnergy. Added GIPSVE_SetLoudspeakerStatus. Updated information about UTF-8 support for all file names. Updated the Error codes list to comply with GIPSVEErrors.h.

		Updated DTMF sub-API chapter with information about non-DTMF telephone events that are now supported. Updated the Trace documentation about the new Trace implementation. Removed GIPSVE_SendRTCP_APP and GIPSVE_SetRTCP_APPCallback from the GIPSVEPTT sub-API. Replaced the GIPSVERTCP_APPCallback class in GIPSVEPTT with the more generic GIPSVERTCPObserver in GIPSVERTP_RTCP. Added GIPSVERTP_RTCP::GIPSVE_SendApplicationDefinedRTCPPacket. Updated Trace filter enumerator names to new Level enumerator. Added tables for file sizes of trace files when different trace filters are used. Updated information about SetSendTOS and SetSendGQoS. Added remark for the file format enumerator clarifying that the FILE_WAV setting must be used for wave files containing wave headers. Replaced InitTimestamp API with SetInitTimestamp and SetInitSeqenceNumber APIs. Added remark to SetSendDestiantion that the DSCP value cannot be set if the send port is specified explicitly. Added a stereo conferencing example to the Volume API section. Updated the GIPSVERTCPObserver class. Renamed member to OnApplicationDataReceived. Added new callback class called GIPSVERTPObserver. Added GIPSVERTP_RTCP::GIPSVE_SetRTPObserver. Added remark to SetSourceFilter and GetSourceFilter that SetLocalReceiver must have been called before using these. Added GIPSVECodec::GIPSVE_Set/GetBandwidthExtensionStatus APIs. Added GIPSVEBase:: GIPSVE_SetNetEQBGNMode APIs. Added new parameter to the GIPSVEBase::GIPSVE_PutOnHold API. It is now possible to control the two directions (input/output) independently. Added documentation to SetSourceFilter about RTCP port filter and modified example. Added new parameter to the GIPSVoiceEngine::Delete API. Now possible to force delete without verifying that all reference counters are zero. Replaced SendExtraRTPPacket and SendExtraRTCPPacket with SendExtraPacket. Updated the trace enumerator documentation to be in line with GIPS_common_types.h. Added remarks to SetTraceFileName() and SetDebugTraceFileName() clarifying that new calls to these functions override old ones, also when the calls are made from difference VE instances. Added new parameter to GIPSVEDTMFCallback::OnDTMFEvent. Added GIPSVE_SetISACMaxRate. Added WinCE / Mobile supported note for SetNetEQBGNMode(), GetNetEQBGNMode(), GetRecPayloadType(), GetRemoteSSRC(), GetRemoteCSRCs(), GetRemoteEnergy(), SetRTPObserver(), SetRTCPObserver(), SendApplicationDefinedRTCPPacket(), SetDTMFDetection(), SendExtraPacket(). Corrected incorrect naming for the SndExtraPacket() function.
3.4.0.0	2009-10-16	Replaced SetIPVersion with EnableIPv6, and GetIPVersion with IPv6IsEnabled. Replaced SendExtraPacket with SendUDPPacket. Added rtcpPort output parameter to the GetSourceFilter API. Updated the Remarks section for SetSendTOS and SetSendGQoS to clarify the new required calling order. Updated the Remarks sections for SetLocalReceiver, SetSendDestination, StartListen and StartSend. Added GIPSVE_SetDTMFPlayoutStatus and GIPSVE_GetDTMFPlayoutStatus. Added GIPSVE_Start/StopRTPDump and GIPSVE_RTPDumpIsActive to the GIPSVERTP_RTCP sub-API. Removed redundant API GIPSVE_GetPTTSessionInfo. Added AECM notes and enumerator. Added GIPSVEVQMonCallback, changed GIPSVE_SetVQMonAlertCalback. Updated the GIPSVE_Set/GetRecPayloadType APIs with information about the added usage of the channel parameter in the codec structure (is related to stereo-playout support). Added newFileOnWrap argument to GIPSVE_SetTraceFileName and GIPSVE_SetDebugTraceFileName. Added new excludeTrace argument to GIPSVE_SetObserver. Added ipv6 parameter to GIPSVE_GetLocalIP . Added note on how to use default communication device on Windows 7 to GIPSVE_SetSoundDevices(). Set deprecated note for GIPSVE_SetConfStatus() and GIPSVE_GetConfStatus() and referred to EC_CONFERENCE mode in GIPSVE_SetECStatus() and NS_CONFERENCE mode in GIPSVE_SetNSStatus(). Added documentation about new modes and behavior to GIPSVE_SetNSStatus(), GIPSVE_SetECStatus() and GIPSVE_SetAGCStatus(). Removed the GIPS_ConfMode enumerators and added the new conference enumerators to GIPS_NSmodes, GIPS_AGCmodes and GIPS_ECmodes.

		Marked GIPSVE_SetG729SilenceThreshold() as OBSOLETE. Added note about filename set to NULL will stop tracing to GIPS_SetDebugTraceFileName() and GIPS_SetTraceFileName(). Updated info about GIPSVEExternalMedia supporting stereo processing for played out audio. Updated information about GIPSVE_StartListen, GIPSVE_StopListen, GIPSVE_SetRecPayloadtype, and GIPSVE_SetExternalTransport. Added stereo-codec example for the GIPSVE_SetRecPayloadtype API. Updated the Error Code section with the latest new error codes.
3.4.1.0	2009-11-17	Removed the stereo-playout parameter from the GIPSVE_SetSoundDevices API.
3.4.2.0	2009-12-03	Added Android platform support notes. Added note about special Create function for Android. Added Android supported note to GIPSVE_Set/GetRTPKeepaliveStatus().
3.4.3.0	2009-12-11	Added RTP_RTCP::GIPSVE_InsertExtraRTPPacket API.
3.4.4.0	2009-12-14	Added Windows CE / Mobile support notes to GIPSVE_Set/Get_DTMFPlayoutStatus, GIPSVE_GetLocalIP, GIPSVE_Start/StopRTPDump, GIPSVE_RTPDumpIsActive, GIPSVE_InsertExtraRTPPacket.
3.4.5.0	2009-12-18	Updated support notes for iPhone.
3.4.6.0	2010-01-20	Not used.
3.4.7.0	2010-01-21	Patch release for Windows, Mac and Linux. No API changes.
3.4.8.0	2010-02-01	Patch release for Windows, Mac and Linux. No API changes.
3.5.0.0	2010-04-06	Added a new chapter called Integration Notes and a section about Windows Audio APIs in VoiceEngine 3.5. Updated all iSAC-related APIs with information about then new super-wideband mode of iSAC. Also updated the Customized Codec Settings section with details about iSAC super-wideband. Added *useForRTCP* parameter in EnableSRTPSend and EnableSRTPReceive functions. Updated the GIPS_AGCmodes enumerator. Added GIPSVEVQE::GIPSVE_Set/GetAGCConfig() and the new GIPS_AGC_config structure. Added GIPSVoiceEngine::SetTraceCallback() API and the GIPSTraceCallback class. Removed the CallbackOnTrace method from the GIPSVoiceEngineObserver class interface. Removed the *excludeTrace* parameter from the GIPSVEBase::GIPSVE_SetObserver() API. Renamed GIPSVEBase:: GIPSVE_SetTraceFileName() to GIPSVoiceEngine::SetTraceFile(). Renamed GIPSVEBase:: GIPSVE_SetDebugTraceFileName() to GIPSVoiceEngine::SetEncryptedTraceFile(). Renamed GIPSVEBase:: GIPSVE_ SetTraceFilter () to GIPSVoiceEngine::SetTraceFilter(). Removed note saying that sending must be active from GIPSVE_SendUDPPacket(). Added Linux and MAC OS X support for the GIPSVE_Set/GetSendTOS() APIs. Fixed "disable" which should read "enable" typo in GIPSVoiceEngine::SetTraceFilter(). Added more detail to the parameter descriptions under GIPSVE_GetEchoMetrics(). Added Mac OS X support to GIPSVE_GetNumOfSoundDevices(). Added new optional *channel* parameter to GIPSVE_StartRecordingMicrophone().

Writing Conventions

This guide uses the following writing conventions:

Convention	Definition
`Code`	Indicates a parameter to which the description is referring.
`Syntax`	Gives the syntax or usage of a function call.
URL	Indicates a jump to an external information source, such as a Web site or a URL.

`Document Link`	Indicates a jump to a section of this document with more information.
NOTE:	Indicates important information that helps to avoid and troubleshoot problems.

Obtaining Documentation

White papers, case studies, test tools, guides, and other documents can be viewed or downloaded from the Global IP Solutions Developer Community Forum at developer.gipscorp.com.

Related Documents

The following guides are related to the GIPS VoiceEngine API Guide:

- GIPS VoiceEngine Mobile for Windows: Integration Notes
- GIPS VoiceEngine Mobile for Symbian: Integration Notes
- GIPS VoiceEngine Mobile for iPhone: Integration Notes
- GIPS VoiceEngine Mobile for Android: Integration Notes

2 Sub-APIs

This chapter gives a short overview of the term sub-API.

The GIPS VoiceEngine PC Lite is a subset of the GIPS VoiceEngine PC, hence many of the features in VoiceEngine are not supported in the Lite version. This chapter lists the supported sub-APIs in VoiceEngine PC Lite.

Overview

The GIPS VoiceEngine API now consists of several different sub-APIs instead of one combined API, where only one sub-API called `GIPSVEBase` is mandatory. All the other APIs adds functionality to the base API but they are not required to set up a standard full duplex G.711 VoIP call.

The main rationale for dividing the API into smaller sub-APIs, or interfaces, is to simplify the usage and to increase the overview of each API.

The table below summarizes the APIs that are currently available in GIPS VoiceEngine.

NOTE: The `GIPSVESymbian` sub-API is only supported on Symbian OS 9.x/S60 3rd Ed. platforms. See GIPS VoiceEngine Mobile for Symbian: Integration Note for more details.

sub-API	Header	Description
GIPSVEBase	GIPSVEBase.h	Enables full duplex VoIP using G.711. NOTE: This API must always be created.
GIPSVECodec	GIPSVECodec.h	Adds non-default codecs (e.g. iLBC, iSAC, G.729 etc.), Voice Activity Detection (VAD) support, and Bandwidth Extension (BWE) functionality.
GIPSVENetwork	GIPSVENetwork.h	Adds external transport, port and address filtering, Windows QoS support and packet timeout notifications.
GIPSVERTP_RTCP	GIPSVERTP_RTCP.h	Adds support for RTCP sender reports, SSRC handling, RTP/RTCP statistics, Forward Error Correction (FEC), RTCP APP, RTP capturing and RTP keepalive.
GIPSVEVQE	GIPSVEVQE.h	Adds support for Noise Suppression (NS), Automatic Gain Control (AGC) and Echo Control (EC). Receiving side VAD is also included.
GIPSVEVolumeControl	GIPSVEVolumeControl.h	Adds speaker volume controls, microphone volume controls, mute support, and additional

		stereo scaling methods.
GIPSVEHardware	GIPSVEHardware.h	Adds sound device handling, CPU load monitoring, external sound card support and device information functions.
GIPSVEDTMF	GIPSVEDTMF.h	Adds telephone event transmission, DTMF tone generation and telephone event detection. (Telephone events include DTMF.)
GIPSVEFile	GIPSVEFile.h	Adds file playback, file recording, file conversion and non-realtime RTP-to-file conversion.
GIPSVEEncryption	GIPSVEEncryption.h	Adds Secure RTP (SRTP) and external encryption/decryption support.
GIPSVEPTT	GIPSVEPTT.h	Adds Push-to-talk (PTT) support, informs about ongoing PTT sessions.
GIPSVEVideoSync	GIPSVEVideoSync.h	Adds RTP header modification support, playout-delay tuning and monitoring.
GIPSVEVQMon	GIPSVEVQMon.h	Adds Telchemy VQMon support, including callback notifications, VoIP metric reports according to RFC 3611, and SIP quality reports.
GIPSVEExternalMedia	GIPSVEExternalMedia.h	Adds support for external media processing and enables utilization of an external audio resource.
GIPSVEG729Extended	GIPSVEG729Extended.h	Adds support for G.729 Annex-B control.
GIPSVECallReport	GIPSVECallReport.h	Adds support for call reports which contains number of dead-or-alive detections, RTT measurements, and Speech, Noise and Echo metrics.
GIPSVESymbian	GIPSVESymbian.h	Adds Symbian-specific functionality. See GIPS VoiceEngine Mobile for Symbian: Integration Note for more details.

More details on how to acquire, use and release sub-APIs are given in Chapter 3

VoiceEngine PC Lite

The following sub-APIs from Chapter 4 are supported in GIPS VoiceEngine Lite:

- GIPSVEBase
- GIPSVECodec
    - The following codecs are supported:

    - G.711,
    - iLBC,
    - Enhanced G.711,
    - iSAC, and
    - iPCM-wb.
- GIPSVENetwork
- GIPSVEVolumeControl

3 Integration Notes

This chapter describes details related to integration of GIPS VoiceEngine on certain platforms.

There is currently only one section and it contains information about the supported audio APIs on Windows.

Windows Audio APIs

Starting with VoiceEngine 3.5, GIPS VoiceEngine is delivered by default with support for Windows Core Audio APIs on Windows Vista and Windows 7. It is detected during initialization if Core Audio is supported or not. If is not supported (e.g. if the platform is Windows XP), GIPS VoiceEngine will automatically revert back to using wave APIs instead. Hence, one common library can be utilized on all Windows platforms and the most suitable audio layer will always used.

Background

The core audio APIs provide the means for audio applications to access audio endpoint devices such as headphones and microphones. The core audio APIs serve as the foundation for higher-level audio APIs such as the Windows multimedia Wave functions, which is still the default internal audio API on Windows XP and Windows 2000.

The Core Audio APIs were introduced in Windows Vista. This is a new set of user-mode audio components provides client applications with improved audio capabilities. The Core Audio APIs have been improved in Windows 7.

GIPS VoiceEngine currently utilizes the following Core Audio APIs:

- Multimedia Device (MMDevice);
- Windows Audio Session API (WASAPI);
- EndpointVolume API.

The audio core APIs are implemented in the Mmdevapi.dll and Audioses.dll system components, both of which run in user mode.

Implementation Details

All deliveries of GIPS VoiceEngine will be built with Microsoft Visual Studio 2005 in combination with a Windows SDK of version 6.0 or higher (see http://en.wikipedia.org/wiki/Microsoft_Windows_SDK for details), to enable Core Audio development. Consequently, the user must also utilize a Windows SDK of version 6.0 or higher, in combination with Visual Studio 2005, when building the final application.

NOTE: the required SDK is not supported in combination with Visual Studio 2003. It means that if a VS 2003 compatible library is required, Windows Core Audio must first be disabled. Please contact GIPS support if such a delivery is needed.

To ensure the best possible audio quality on Windows Vista and Windows 7, GIPS Voice Engine 3.5 also supports Multimedia Class Scheduler Service (MMCSS) in combination with the Core Audio API. The MMCSS boosts the priority of threads that are working on high-priority multimedia tasks.

Windows Core Audio APIs are based on the Microsoft Component Object Model (COM) and GIPS VoiceEngine uses COM objects in a thread safe architecture.

The default COM usage in GIPS VoiceEngine is that each thread that uses COM, and every object that those threads create, is assigned to a multithreaded apartment (MTA), hence: `CoInitializeEx(NULL, COINIT_MULTITHREADED)` is called for each thread. In addition, each call to `CoInitializeEx` is matched with a corresponding call to `CoUninitialize()` at termination.

If the user of GIPS VoiceEngine has already initialized COM for a single-threaded apartment model by calling `CoInitializeEx(NULL, COINIT_APARTMENTTHREADED)`, a COM initialization conflict will be detected during initialization and GIPS avoids calling `CoInitializeEx(NULL, COINIT_MULTITHREADED)`. In addition, `CoUninitialize()` is not called at termination.

Known Limitations

Even if the application runs on Windows Vista or Windows 7, it can still happen that GIPS VoiceEngine uses the old Wave APIs. The decision to revert back to Wave APIs is taken during the initialization (see `GIPSVEBase::GIPSVE_Init()`) and the most common reason for failing to support Core Audio is that the user has selected an endpoint device with a default audio format that is not supported by GIPS VoiceEngine.

The preferred combination of sample rate, bit depth and number of channels is 48000 Hz, 16 bits and 2 channels but the following sample rates are also supported: 44100, 16000, 96000, 32000 and 8000. An example of a setting that is not supported is 192000 Hz and 24 bits.

There is currently no explicit API in GIPS VoiceEngine to determine if Core Audio or Wave audio is used internally. However, it is possible to find out in an indirect way by calling `GIPSVEHardware::GIPSVE_GetRecordingDevName()` or `GIPSVEHardware::GIPSVE_GetPlayoutDevName()`. The `strGuidUTF8` parameter in both these APIs will only be a unique GUID string on Windows Vista and Windows 7 if Core Audio is active. For all other cases, `strGuidUTF8` is only a copy of the product name (max size is 32 characters) contained in the `strNameUTF8` output parameter.

4 API Overview

This chapter summarizes how to allocate resources, acquire and use sub-APIs, and how to release resources in a correct way. It also describes how to enable callbacks for GIPS trace messages.

VoiceEngine Classes, Structures and Enumerators

This section describes classes and structures that are common for the GIPSVoiceEngine class and all its all sub-APIs. They are all declared in the files GIPS_common_types.h and GIPSVECommon.h.

Struct GIPS_CodecInst

The VoiceEngine holds information about each codec it supports in this format.

Syntax

```
struct GIPS_CodecInst
{
    int pltype;
    char plname[16];
    int plfreq;
    int pacsize;
    int channels;
    int rate;
};
```

Parameters

pltype	The payload type. Payload type can be set in the structure before it is passed to `GIPSVEBase::GIPSVE_SetSendCodec()`.
plname	The MIME name.
plfreq	The sampling-frequency of the codec.
pacsize	The number of samples to be sent in each packet. The codec structure holds a default packet size value in samples, even though supported codecs can handle other sizes as well. Packet size can be set in the structure before it is passed to `GIPSVEBase::GIPSVE_SetSendCodec()`.

channels	The number of audio channels (1=mono, 2=stereo). The channels parameter may be omitted in the SDP message if the number of channels is one and if no additional parameters are needed.
rate	The codec bit rate in bits per second. -1 corresponds to channel-adaptive rate which is supported for the iSAC codec.

Remarks

Refer to Customizing Codec Settings for more information.

Example

A codec defined in the SDP header `a=rtpmap:97 iPCMWB/16000/1` has:

- `pltype` = 97
- `plname` = iPCMWB
- `plfreq` = 16000
- `channels` = 1

Class InStream

This is a base class for reading data from, for example, a file. It is up to the VoiceEngine user to implement a derived class with the `Read()` function overridden.

```
class InStream
{
    virtual int Read(void *buf, int len) = 0;
};
```

InStream::Read

This function should read `len` bytes and put these in `buf`. If less than `len` bytes remain when the function is called, all the remaining bytes should be put into `buf`.

Syntax

```
int Read(void *buf, int len);
```

Parameters

buf	[out] A pointer to an array into which the read data should be copied.
len	[in] The number of bytes to read.

Return Values

n	The return value specifies the number of bytes that were put into `buf`.
–1	An error occurred.

Class OutStream

This is a base class for writing data to, for example, a file. It is up to the VoiceEngine user to implement a derived class with the `Write()` function overridden.

```
class OutStream
{
    virtual bool Write(void *buf, int len) = 0;
};
```

InStream::Write

This function should write `len` bytes from `buf` to the desired location.

Syntax

```
bool Write(void *buf, int len);
```

Parameters

buf	[in] A pointer to an array containing the data to be written.
len	[in] The number of bytes to write.

Return Values

true	The call was successful.
false	The call failed.

Class GIPS_transport

This class declares an abstract interface for a user definable external transport protocol. It is up to the VoiceEngine user to implement a derived class which overrides `SendPacket()` and `SendRTCPPacket()`.

```
class GIPS_transport
{
public:
    virtual int SendPacket(int channel, const void* data, int len) = 0;
    virtual int SendRTCPPacket(int channel, const void* data, int len) = 0;
};
```

GIPS_transport::SendPacket

This method will be called by the VoiceEngine for each block of data that is recorded, encoded and packetized into RTP packets.

Syntax

```
int SendPacket(int channel, const void* data, int len);
```

Parameters

channel	[in] The channel ID number.
data	[in] Pointer to data buffer which contains the RTP packet to be transmitted.
len	[in] Length, in number of bytes, of the `data` buffer.

Return Values

The number of bytes actually transmitted (normally the same as `len`) or -1 if an error occurred.

GIPS_transport::SendRTCPPacket

This method will be called by the VoiceEngine for each block of data that is recorded, encoded and packetized into RTCP packets.

Syntax

```
int SendRTCPPacket(int channel, const void* data, int len);
```

Parameters

channel	[in] The channel ID number.
data	[in] Pointer to data buffer which contains the RTCP packet to be transmitted.
len	[in] Length, in number of bytes, of the `data` buffer.

Return Values

The number of bytes actually transmitted (normally the same as `len`) or -1 if an error occurred.

Remarks

The standard configuration of GIPS VoiceEngine uses RTP/UDP/IP to transmit data over the network, but it does also support usage of an external transport protocol, if configured in that particular mode. In the external transportation-mode, sending and receiving packets from the network must be handled by the user outside of the VoiceEngine.

This class provides an additional interface that enables the VoiceEngine to call a send-function once a block of speech has been recorded, encoded and packetized and a call to pass the packet received from the network to the VoiceEngine.

The VoiceEngine will deliver RTP/RTCP packets to the `SendPacket()/SendRTCPPacket()` functions and expects to receive the RTP/RTCP packets from the network. The information is packetized in RTP/RTCP-format because information such as payload type and sequence number is vital to make a correct decoding of the data.

This class must be implemented by the user, which then will allow VoiceEngine to call the send function once a block of data is recorded, encoded and packetized.

Refer to `GIPSVENetwork::GIPSVE_SetExternalTransport()` for more information.

Class GIPS_encryption

VoiceEngine provides an interface that enables you to add a custom encryption scheme to the RTP stream. The `GIPS_encryption` class is a callback class for adding such an encryption scheme.

The VoiceEngine user should override the methods in a derived class.

Syntax

```
class GIPS_encryption
{
public:
    void encrypt(int channel_no, unsigned char *in_data, unsigned char *out_data,
                int bytes_in, int *bytes_out);
    void decrypt(int channel_no, unsigned char *in_data, unsigned char *out_data,
                int bytes_in, int * bytes_out;
    void encrypt_rtcp(int channel_no, unsigned char *in_data, unsigned char
                *out_data, int bytes_in, int *bytes_out);
    void decrypt_rtcp(int channel_no, unsigned char *in_data, unsigned char
                *out_data, int bytes_in, int *bytes_out);
};
```

Parameters

channel_no	[in] The channel ID number.
in_data	[in] A pointer to an array containing the input data.
out_data	[out] A pointer to an array to which the output data should be copied.
bytes_in	[in] The size of the array pointed to by `in_data` in bytes.
bytes_out	[out] The size of the output array in bytes should be copied to the pointee.

Remarks

The `encrypt` and `encrypt_rtcp` functions will be called for every RTP and RTCP packet respectively that is ready to be sent. The entire packet (including header) will be pointed to by `in_data`. If the RTP header should not be encrypted, the first 12 bytes should be left un-touched. The encrypted packet should be copied to the `out_data` array, and the length to `bytes_out`.

1500 bytes are allocated internally for the `out_data` array, hence it is not allowed to write outside this boundary.

The same methodology is used for the decrypt calls.

The derived instance is installed with `GIPSVEEncryption::GIPSVE_InitEncryption()`.

Class GIPSTraceCallback

This class declares an abstract interface for a user definable external trace protocol. It is up to the VoiceEngine user to implement a derived class which overrides the `Print()` method.

```
class GIPSTraceCallback
{
public:
```

```
virtual void Print(const GIPS::TraceLevel level, const char *traceString, const
   int length) = 0;
};
```

GIPSTraceCallback::Print

This method is called for each non-encrypted trace message produced by VoiceEngine (or any other active GIPS Engine, e.g. GIPS VideoEngine) if callback traces has been enabled using `GIPSVoiceEngine::SetTraceCallback()`.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages which can affect the voice quality in a negative way.

Syntax

```
void Print(const GIPS::TraceLevel level, const char *traceString, const int
   length);
```

Parameters

level	[out] Enumerator specifying the trace message type.
traceString	[out] Pointer to character buffer which contains the trace message string sent to the observer. The string is null-terminated.
length	[out] Length, in number of characters, of the `traceString`. The length includes the null-termination character.

Enumerator GIPS::TraceLevel

This enumerator specifies what type of trace filter to use.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages which can affect the voice quality in a negative way.

Syntax

```
namespace GIPS
{
    enum TraceLevel
    {
        TR_NONE          = 0x0000,
        TR_STATE_INFO    = 0x0001,
        TR_WARNING       = 0x0002,
        TR_ERROR         = 0x0004,
        TR_CRITICAL      = 0x0008,
        TR_APICALL       = 0x0010,
        TR_MODULE_CALL   = 0x0020,
```

```
        TR_DEFAULT        = 0x00FF,

        TR_MEMORY         = 0x0100,
        TR_TIMER          = 0x0200,
        TR_STREAM         = 0x0400,

        // everything bellow will be encrypted
        // and is used for GIPS debug purposes
        TR_DEBUG          = 0x0800,
        TR_INFO           = 0x1000,
        TR_CUSTOMER       = 0x2000,

        TR_ALL            = 0xFFFF
    };
};
```

Enumerators

TR_NONE	Disables all trace messages.
TR_STATE_INFO	Used for status messages, such as “incoming bit rate is 100 kbps”, or “function X is now active”
TR_WARNING	Used for warning messages, such as “CPU load is too high”, or “function is already active”.
TR_ERROR	Used for error messages, such as “invalid parameter”, or “unable to open file”.
TR_CRITICAL	Used for critical messages, such as “soundcard failed to play out data”.
TR_APICALL	Used for all GIPS API calls, such as “GIPSVE_Init()”.
TR_MODULE_CALL	Used for GIPS internal module calls; will lead to a very large amount of trace messages.
TR_DEFAULT	Used for default, non encrypted messages.
TR_MEMORY	Used for memory debug information.
TR_TIMER	Used for timing debug information; can lead to large amount of trace messages.
TR_STREAM	Used for audio stream debug information; will lead to a very large amount of trace messages.
TR_DEBUG	[encrypted] Internal GIPS debug information; can lead to large amount of trace messages.
TR_INFO	[encrypted] Internal GIPS debug information; can lead to large amount of trace messages.

TR_CUSTOMER	[encrypted] Internal GIPS debug information.
TR_ALL	Enables all trace messages.

Remarks

It is possible to combine several different values into one singe filter using the logical OR (|) operator. Example: GIPS::TR_STATE_INFO | GIPS::TR_WARNING | GIPS::TR_ERROR | GIPS::TR_CRITICAL.

Declared in the GIPS_common_types.h header file.

Note that these enumerators are within the GIPS namespace.

Enumerator GIPS_StereoChannel

This enumerator specifies what stereo mode to use.

Syntax

```
enum GIPS_StereoChannel
{
    GIPS_StereoLeft = 0,
    GIPS_StereoRight,
    GIPS_StereoBoth
;
```

Enumerators

GIPS_StereoLeft	Select the left channel.
GIPS_StereoRight	Select the right channel.
GIPS_StereoBoth	Select both left and right channels.

Remarks

Declared in the GIPSVECommon.h header file.

Class GIPSVoiceEngine

This section describes how to allocate and release resources for the GIPS VoiceEngine using factory methods in the `GIPSVoiceEngine` class. It also lists the APIs which are required to enable file tracing and/or traces as callback messages.

The main steps required to create and release the VoiceEngine are:

1. Call the static factory method `GIPSVoiceEngine::Create()` to acquire pointer to a `GIPSVoiceEngine` object.
2. Release the VoiceEngine resources by calling `GIPSVoiceEngine::Delete()`.

NOTE: The acquired `GIPSVoiceEngine` pointer cannot be used to access any API methods. It must first be converted to a so called sub-API pointer. Refer to the How to Acquire and Release VoiceEngine Sub-APIs section in this chapter for more details.

When the VoiceEngine is created, it is possible to enble trace messages for debugging purposes. The main steps are:

1. If no other GIPS Engine is already using the trace class, call `GIPSVoiceEngine::SetTraceFile()` to enable file tracing.
2. Specify a suitable trace filter (affects the amount of generated information) by calling `GIPSVoiceEngine::SetTraceFilter()`.
3. Use the VoiceEngine.
4. If no other GIPS Engine is still using the trace class, disable trace messages by calling `GIPSVoiceEngine::SetTraceFile()` with NULL as input parameter.
5. Analyze the trace output stored on file.

NOTE: all trace messages in GIPS Engine products are generated by a singleton instance. Hence, using this API will affect all currently active GIPS Engines. As an example: if two VoiceEngine instances are created, it is up to the user to ensure that only one of the instances calls this API to avoid conflicts.

The `GIPSVoiceEngine` class is declared in GIPSVEBase.h:

```
class GIPSVoiceEngine
{
public:
    static GIPSVoiceEngine* Create();
    static bool Delete(GIPSVoiceEngine*& voiceEngine);
    static int SetTraceFilter(const unsigned int filter);
    static int SetTraceFile(const char* fileNameUTF8, const bool addFileCounter =  false);
    static int SetEncryptedTraceFile(const char* fileNameUTF8, const bool addFileCounter = false);
    static int SetTraceCallback(GIPSTraceCallback* callback);
};
```

GIPSVoiceEngine::Create

Creates a `GIPSVoiceEngine` object, which can then be used to acquire sub-APIs.

Syntax

```
static GIPSVoiceEngine* Create();
```

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVoiceEngine` object.

If the function fails, the return value is NULL.

Remarks

The acquired `GIPSVoiceEngine` pointer cannot be used to access any API methods. It must first be converted to a so called sub-API pointer. See example code below and the section called How to Acquire and Release VoiceEngine Sub-APIs in this chapter for more details.

NOTE: There is a special Create API function for Android. Please see the Android Integration Notes document.

Example Code

```
#include "GIPSVEBase.h"

// create the VoiceEngine
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();

// acquire, use and release the sub-API
GIPSVEBase* base = GIPSVEBase::GIPSVE_GetInterface(ve);
base->GIPSVE_Init();
base->GIPSVE_Terminate();
base->GIPSVE_Release();

// delete the VoiceEngine
GIPSVoiceEngine::Delete(ve);
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVoiceEngine::Delete

Deletes a created `GIPSVoiceEngine` object and releases the utilized resources.

Syntax

```
static bool Delete(GIPSVoiceEngine*& voiceEngine, bool ignoreRefCounters =
  false);
```

Parameters

voiceEngine
[in] Pointer reference to an already created `GIPSVoiceEngine` object.

ignoreRefCounters
[in_opt] If set to `false`, all reference counters must be zero to enable a valid release of the allocated resources. When set to `true`, a release of all resources allocated by the VE is performed without checking the reference counter state.

Return Values

true
The call was successful. All resources are released.

false
At least one sub-API has not been released properly. If this is the case, memory will leak. It is possible to override this state by setting `ignoreRefCounters` to `true`.

Remarks

A successful call to this function also modifies the input parameter and sets the pointer to NULL.

It is recommended to ensure that the reference count for each sub-API is zero before calling this method. The `ignoreRefCounters` flag should only be set to `true` if it is not possible to ensure that the number of calls to `GetInterface()` matches the number of calls to `Release()` for a given sub-API.

Example Code

```
// Example #1 – correct usage where the sub-API is released properly
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();
GIPSVEBase* base = GIPSVEBase::GIPSVE_GetInterface(ve);
base->GIPSVE_Release();
GIPSVoiceEngine::Delete(ve);  // returns true (recommended usage)

// Example #2 – incorrect usage where the sub-API is not properly released
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();
GIPSVEBase* base = GIPSVEBase::GIPSVE_GetInterface(ve);
GIPSVoiceEngine::Delete(ve);  // returns false, must release GIPSVEBase first

// Example #3 – correct usage even if the sub-API is not properly released
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();
GIPSVEBase* base = GIPSVEBase::GIPSVE_GetInterface(ve);
GIPSVoiceEngine::Delete(ve, true);  // forces delete and returns true (not recommended)
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVoiceEngine::SetTraceFilter

Specifies the amount and type of trace information, which will be created by the GIPS VoiceEngine.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
static int SetTraceFilter(const unsigned int filter);
```

Parameters

filter [in] Sets the filter type. See the `GIPS::TraceLevel` enumerator for filter details.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

At least one GIPS Engine must have been created before calling this API to ensure that a singleton trace implementation exists.

Default filter type is TR_STATE_INFO | TR_WARNING | TR_ERROR | TR_CRITICAL | TR_APICALL.

To enable all traces, use the TR_ALL filter type.

To disable all traces, use the TR_NONE filter type.

Valid trace file names must have been set before this filter has any effect. See `GIPSVoiceEngine::SetTraceFile()` and `GIPSVoiceEngine::SetEncryptedTraceFile()` for details.

The filtered non-encrypted trace messages will also be sent as callback messages if a `GIPSTraceCallback` instance has been installed by the `GIPSVoiceEngine::SetTraceCallback()` API.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVoiceEngine::SetTraceFile

Sets the name of the trace file and enables non-encrypted trace messages.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
static int SetTraceFile(const char* fileNameUTF8, const bool addFileCounter =
  false);
```

Parameters

fileNameUTF8 [in] Pointer to a zero-terminated and UTF-8 encoded character string, which contains the name of the trace file. If set to NULL the tracing to a previously set file will stop and the file will be closed.

addFileCounter [in_opt] If set to true, the trace file will not wrap when the file size limit is reached. Instead, a new file will be created and the following messages will be written to that file. A number extension of the form "_#" will be added to the file name.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

At least one GIPS Engine must have been created before calling this API to ensure that a singleton trace implementation exists.

The type and amount of trace information is set by the `GIPSVoiceEngine::SetTraceFilter()` API. See the `GIPS::TraceLevel` enumerator for filter details. Note that the encrypted traces will not be printed to the file generated by this API even if those filters are set.

The filtered non-encrypted trace messages will also be sent as callback messages if a `GIPSTraceCallback` instance has been installed by the `GIPSVoiceEngine::SetTraceCallback()` API.

Calling this function will override any previous `GIPSVoiceEngine::SetTraceFile()` calls. VoiceEngine will stop writing to the old file and write to the new one instead. This applies when the calls are made from different instances of VoiceEngine or other GIPS Engines as well.

The non-encrypted trace information corresponds to the following filter:
GIPS::TR_STATE_INFO | GIPS::TR_WARNING |GIPS::TR_ERROR |GIPS::TR_CRITICAL |GIPS::TR_APICALL | GIPS::TR_MODULE_CALL, or GIPS::TR_ALL.

The TR_MODULE_CALL traces are not printed per default since they generate a lot of trace information. See the file size table below. They can be enabled by using the `GIPSVoiceEngine::SetTraceFilter()` API.

The generated trace massages may differ in formatting since the trace implementation is in a transition phase between old and new formatting.

Filter	**Approximate file size for 1 min of tracing**
default	44 kB
GIPS::TR_ALL	790 kB

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVoiceEngine::SetEncryptedTraceFile

Sets the name of the debug trace file and enables encrypted (internal) trace messages.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
static int SetEncryptedTraceFile(const char* fileNameUTF8, const bool
  addFileCounter = false);
```

Parameters

fileNameUTF8 [in] Pointer to a zero-terminated and UTF-8 encoded character string, which contains the name of the trace file. If set to NULL the tracing to a previously set file will stop and the file will be closed.

addFileCounter [in_opt] If set to true, the trace file will not wrap when the file size limit is reached. Instead, a new file will be created and the following messages will be written to that file. A number extension of the form "_#" will be added to the file name.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

At least one GIPS Engine must have been created before calling this API to ensure that a singleton trace implementation exists.

Calling this function will override any previous `GIPSVoiceEngine::SetEncryptedTraceFile()` calls. VoiceEngine will stop writing to the old file and write to the new one instead. This applies when the calls are made from different instances of VoiceEngine or other GIPS Engines as well.

The type and amount of trace information is set by the `GIPSVoiceEngine::SetTraceFilter()` API. See the `GIPS::TraceLevel` enumerator for filter details.

All trace message types listed in the `GIPS::TraceLevel` enumerator can be printed to the file generated by this API and all message types will be encrypted.

Filter	Approximate file size for 1 min of tracing
GIPS::TR_ALL	4 MB
All filters except GIPS::TR_MODULE_CALL	3.2 MB
All filters except GIPS::TR_STREAM	1.1 MB

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVoiceEngine::SetTraceCallback

Installs the `GIPSTraceCallback` implementation to ensure that the VoiceEngine user receives callbacks for generated trace messages.

Syntax

```
static int SetTraceCallback(GIPSTraceCallback* callback);
```

Parameters

callback [in] An instance of the `GIPSTraceCallback` derived class.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` .

Remarks
At least one GIPS Engine must have been created before calling this API to ensure that a singleton trace implementation exists.

It is possible to disable trace callbacks by setting the `callback` parameter to NULL.

Enabling trace callbacks does not conflict with any ongoing trace activity to a specified trace file. See `GIPSVEBase::GIPSVE_SetTraceFileName()` for more details.

See `GIPSVEBase::GIPSVE_SetTraceFilter()` for details on how to specify the amount and type of trace information.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEBase.h

How to Acquire and Release VoiceEngine Sub-APIs

This section describes how to acquire and release sub-APIs given an already created `GIPSVoiceEngine` instance. The main steps are (taking the mandatory `GIPSVEBase` sub-API as an example):

1. Use an existing `GIPSVoiceEngine` pointer as input to the static `GIPSVEBase::GIPSVE_GetInterface()` method to acquire a pointer to the concrete sub-API.
2. Use the `GIPSVEBase` sub-API pointer to call any of the API methods declared in GIPSVEBase.h.
3. Release the sub-API by calling `GIPSVEBase::GIPSVE_Release()`.

The VoiceEngine Sub-APIs table in Chapter 2 lists all the supported sub-APIs in GIPS VoiceEngine, and the described procedure above can be repeated for all APIs in this table.

NOTE: The `GetInterface()` method for `GIPSVEBase` is prefixed by GIPSVE_ to make its name unique. The `GetInterface()` method for all other sub-APIs is called `GetInterface()` only (e.g. `GIPSVECodec::GetInterface()`). The same rules apply to the `Release()` methods, where `GIPSVEBase` has a unique name called `GIPSVEBase::GIPSVE_Release()`.

Example
The following example summarizes the main steps that are needed to acquire, use and then release two different GIPS VoiceEngine sub-APIs. Error handling is omitted.

```
#include "GIPSVEBase.h"
#include "GIPSVEVQE.h"

// allocate resources
```

```
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();

// acquire sub-APIs
GIPSVEBase* base = GIPSVEBase::GIPSVE_GetInterface(ve);  // GIPSVE_ prefix
GIPSVEVQE* vqe = GIPSVEVQE::GetInterface(ve);

// use sub-APIs
base->GIPSVE_Init();
vqe->GIPSVE_SetNSStatus(true, NS_HIGH_SUPPRESSION);
base->GIPSVE_Terminate();

// release all sub-APIs
base->GIPSVE_Release();  // GIPSVE_ prefix
vqe->Release();

// free resources
GIPSVoiceEngine::Delete(ve);
```

5 API Reference

This chapter serves as a reference guide for the GIPS VoiceEngine sub-APIs. All member functions in each sub-API are described in terms of syntax, return values, remarks, code examples and requirements.

For each API method, a requirement table is given. It lists the following type of requirements:

- Supported platforms (e.g. Windows, MAC OS X, Linux)
- VoiceEngine configuration (standard or special configuration)
- Header file (name of header file which declares the sub-API)

NOTE: Always check for what combination of platform and VoiceEngine configuration each API is supported.

GIPSVEBase

The `GIPSVEBase` API is the only mandatory API and it is not possible to build a GIPS VoIP client without it. In short, `GIPSVEBase` enables full duplex VoIP sessions via RTP using G.711 (mu-Law or A-Law). Additionally, this API includes:

- Authentication (for DLL builds only).
- Initialization and termination.
- Trace information on text files or via callbacks.
- Multi-channel support (mixing, sending to multiple destinations etc.).
- Call setup (port and address) for receiving and sending sides.
- Start/Stop of full duplex VoIP streams using G.711.
- Conferencing.

To support other codecs than G.711, the `GIPSVECodec` sub-API must be utilized.

Enumerator GIPS_NetEQModes

This enumerator specifies what type of playout format to use when the NetEQ playout format is modified. It is utilized by the `GIPSVE_SetNetEQPlayoutMode()` and `GIPSVE_GetNetEQPlayoutMode()` APIs.

Syntax

```
enum GIPS_NetEQModes
{
```

```
    NETEQ_DEFAULT = 0,
    NETEQ_STREAMING = 1,
    NETEQ_FAX = 2
};
```

Enumerators

NETEQ_DEFAULT This is the standard mode for VoIP calls. The trade-off between low delay and jitter robustness is optimized for high-quality two-way communication. NetEQ's packet loss concealment and signal processing capabilities are fully employed.

NETEQ_STREAMING In the case of one-way communication – for instance a passive conference participant, a webinar, or a streaming application – this mode can be engaged to improve the jitter robustness at the cost of increased delay. The same set of tools and algorithms as in the NETEQ_DEFAULT mode. The effective delay increase is dependent on the network jitter characteristics.

NETEQ_FAX The fax mode is optimized for decodability of fax signals rather than for perceived audio quality. When this mode is selected, NetEQ will do as few delay changes as possible, trying to maintain a high and constant delay. Meanwhile, the packet loss concealment efforts are reduced.

Enumerator GIPS_NetEQBGNModes

This enumerator specifies what type of background noise (BGN) mode to use when the NetEQ BGN format is modified. It is utilized by the `GIPSVE_SetNetEQBGNMode()` and `GIPSVE_GetNetEQPlayoutMode()` APIs.

If the incoming RTP stream stops abnormally (i.e., not during VAD/DTX silence periods) NetEQ will at first try to extrapolate the latest speech signal to produce an output signal while waiting for the stream to resume. If the interruption last for a longer time, the synthetic speech extrapolation can be replaced with a background noise, generated internally from parameters estimated previously from the incoming signal.

Syntax

```
enum GIPS_NetEQBGNModes
{
    GIPS_BGN_ON = 0,
    GIPS_BGN_FADE = 1,
    GIPS_BGN_OFF = 2
};
```

Enumerators

GIPS_BGN_ON Background noise is generated as long as output audio is extracted from the output side of GIPS NetEQ. This is the default mode.

GIPS_BGN_FADE The background noise is faded to zero (complete silence) after a few seconds.

GIPS_BGN_OFF Background noise is not used at all. In this mode, silence is produced after speech extrapolation has faded.

Enumerator GIPS_LinuxAudio

This enumerator specifies what type of Linux audio driver to use. It is utilized by the `GIPSVE_Init()` API.

Syntax

```
enum GIPS_LinuxAudio
{
   LINUX_AUDIO_OSS = 0,
   LINUX_AUDIO_ALSA
};
```

Enumerators

LINUX_AUDIO_OSS	Open Sound System (OSS) Linux audio driver.
LINUX_AUDIO_ALSA	Advanced Linux Sound Architecture (ALSA).

Enumerator GIPS_OnHoldModes

This enumerator specifies which direction(s) should be affected by the on-hold operation. It is utilized by the `GIPSVE_PutOnHold()` API.

Syntax

```
enum GIPS_OnHoldModes
{
    HOLD_SEND_AND_PLAY = 0,
    HOLD_SEND_ONLY,
    HOLD_PLAY_ONLY
};
```

Enumerators

HOLD_SEND_AND_PLAY	Put both sending and playing directions in on-hold state. This is the default state.
HOLD_SEND_ONLY	Put only the sending directions in on-hold state.
HOLD_PLAY_ONLY	Put only the playing directions in on-hold state.

Class GIPSVoiceEngineObserver

This class declares an abstract interface for a user definable observer mechanism. It is up to the VoiceEngine user to implement a derived class which implements the observer class. The observer is installed and activated by the `GIPSVE_SetObserver()` API.

NOTE: Always ensure that the callback functions are kept as short as possible to ensure that additional callbacks are not delayed.

```
class GIPSVoiceEngineObserver
{
public:
   virtual void CallbackOnError(const int errCode, const int channel) = 0;
};
```

GIPSVoiceEngineObserver::CallbackOnError

This method will be called after the occurrence of any runtime error code when the observer interface has been installed and activated using the `GIPSVE_SetObserver()` API.

Syntax

```
void CallbackOnError(const int errCode, const int channel);
```

Parameters

errCode	[out] The runtime error code sent to the observer.
channel	[out] The channel ID number. When a channel number is not applicable, -1 is given.

Remarks

See AppendixB: Runtime Error Codes for possible runtime errors.

GIPSVE_GetInterface

Retrieves a pointer to the `GIPSVEBase` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEBase* GIPSVE_GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEBase` interface.

If the function fails, the return value is `NULL`.

Remarks

See `GIPSVoiceEngine::Create()` for details on how to create the `GIPSVoiceEngine` object.

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `GIPSVE_Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

```
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();
GIPSVEBase* base = GIPSVEBase::GIPSVE_GetInterface(ve);
if (NULL != base)
```

```
{
    // access valid GIPSVEBase methods within this scope
}
else
{
    // take actions given the invalid interface pointer
}
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_Release

Releases the `GIPSVEBase` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int GIPSVE_Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

The number of calls to `GIPSVE_Release()` should always match the number of calls to `GIPSVE_GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `GIPSVE_Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `GIPSVE_Release()` does not match the number of calls to `GIPSVE_GetInterface()`.

Example Code

```
GIPSVoiceEngine* ve = GIPSVoiceEngine::Create();

GIPSVEBase* base1 = GIPSVEBase::GIPSVE_GetInterface(ve);  // ref. counter = 1
GIPSVEBase* base2 = GIPSVEBase::GIPSVE_GetInterface(ve);  // ref. counter = 2

int count(-1);
count = base1->GIPSVE_Release();  // count = 1 => not OK to delete yet
count = base2->GIPSVE_Release();  // count = 0 => OK to delete
count = base2->GIPSVE_Release();  // count = -1 => error but still OK to delete

if (false == GIPSVoiceEngine::Delete(ve))
```

```
{
    // delete failed, probably due to unreleased sub API(s) => memory will leak
}
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_SetObserver

Installs the observer class to enable runtime error control.

```
int GIPSVE_SetObserver(GIPSVoiceEngineObserver& observer, bool clear = false);
```

Parameters

observer [in] An instance of the `GIPSVoiceEngineObserver` derived class.

clear [in] Set this flag to true to clear the callback mechanism, and false otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

The observer callback methods are generated within a critical section. It is therefore recommended not to call any VoiceEngine APIs within the user-implemented callback functions. Instead, ensure that the callback functions are kept as short as possible to prevent possible deadlocks.

See Section the Class GIPSVoiceEngineObserver section for more information.

See Appendix B: Runtime Error Codes for a list of possible runtime error codes.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_Authenticate

Authenticates usage of the GIPS VoiceEngine, given that the VoiceEngine has been delivered as a DLL.

Syntax

```
int GIPSVE_Authenticate(const char* key, unsigned int length);
```

Parameters

key [in] A pointer to an array containing the authentication string.

length [in] The length, in 8-bit characters, of the key string.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVE_LastError().

Remarks

If VoiceEngine is delivered as a DLL, the DLL needs to be unlocked before any call to VoiceEngine can be done. The purpose of this is to avoid any unauthorized usage of the VoiceEngine DLL. A customer-unique password string is delivered together with the DLL and this string is used to unlock the DLL.

NOTE: The password string MUST be embedded in the calling exe-file, and not stored in any resource-file or registry key.

Requirements

Supported platforms	Windows (incl. CE/Mobile), Symbian (S60 3rd Ed.), Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_Init

Initiates all common parts of the VoiceEngine; e.g. all encoders/decoders, the sound card and core receiving components.

Syntax

```
int GIPSVE_Init(int month = 0, int day = 0, int year = 0, bool recordAEC = false,
  GIPS_LinuxAudio audiolib = LINUX_AUDIO_ALSA);
```

Parameters

month [in_opt] The month at which the VE expires. If the VE is not time-limited (default), month should be set to 0.

day [in_opt] The day at which the VE expires. If the VE is not time-limited (default), day should be set to 0.

year [in_opt] The year at which the VE expires. If the VE is not time-limited (default), year should be set to 0.

recordAEC [in_opt] If set to true, VE will record the echo canceller data to files, to make it possible to analyze the echo canceller behavior offline. Enable this mode only after being advised to do so by a GIPS engineer.

audiolib [in_opt] A GIPS_LinuxAudio enumerator that sets what type of Linux audio driver to use. This parameter is only utilized for Linux platforms.

Return Values

The return value is 0 if the function succeeds or if a "safe/soft" error happens during initialization. If so happens, GIPSVE_LastError() will return a non-zero result, which can be seen as a warning instead of an error.

If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

`GIPSVE_LastError()` can be non-zero even if the returned value from `GIPSVE_Init()` is zero. It is therefore recommended to always check the error code after initialization.

Several core API functions require that this function has been called first. If `GIPSVE_Init()` has not been called, most APIs will return -1 and `GIPSVE_LastError()` will return VE_NOT_INITED.

If you have a time-limited VoiceEngine library, you need to provide the expiry information in the month, day, and year values here. If the library will expire on Jan 21 2009, for example, set `month` = 1, `day` = 21 and `year` = 2009. This prevents the deployment of a time-limited library.

Compatibility with PulseAudio on Linux is ensured when using ALSA as audio lib. Refer to PulseAudio web page for more information about using ALSA applications with PulseAudio.

Example Code

```
base->GIPSVE_Init();                    // default usage, or
base->GIPSVE_Init(1, 21, 2009);         // time limited to Jan 21 2009
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_Terminate

Terminates all VoiceEngine functions and kills the VoiceEngine instance.

Syntax

```
int GIPSVE_Terminate();
```

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_MaxNumOfChannels

Retrieves the maximum number of channels that can be created in this particular build of GIPS VoiceEngine.

Syntax

```
int GIPSVE_MaxNumOfChannels();
```

Return Values

The return value is always a positive integer which corresponds to the maximum number of supported channels.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_CreateChannel

Creates a new channel and allocates the required resources for it.

Syntax

```
int GIPSVE_CreateChannel();
```

Return Values

If the function succeeds, the return value is the channel ID that can be used in all function calls requiring a channel ID as input. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

It is not possible to create a new channel before `GIPSVE_Init()` has been called successfully.

Example Code

```
base->GIPSVE_Init();
int chID = base->GIPSVE_CreateChannel();
if (chID != -1)
{
    base->GIPSVE_SetLocalReceiver(chID, 12345);
    base->GIPSVE_StartListen(chID);
    base->GIPSVE_StartPlayout(chID);

    // …

    base->GIPSVE_DeleteChannel(chID);
}
base->GIPSVE_Terminate();
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard

Header	Declared in GIPSVEBase.h

GIPSVE_DeleteChannel

Deletes an existing channel and releases the utilized resources.

Syntax

```
int GIPSVE_DeleteChannel(int channel);
```

Parameters

channel [in] The channel number ID.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Example Code

See `GIPSVE_CreateChannel()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_SetLocalReceiver

Defines the local receiver port and address for a specified channel number.

Syntax

```
int GIPSVE_SetLocalReceiver(int channel, int port, int RTCPport = GIPS_DEFAULT,
   const char* ipaddr = NULL, const char* multiCastAddr = NULL);
```

Parameters

channel [in] The channel number ID.

port [in] RTP/UDP port number to receive packets on. This port is also the source port for sending, i.e., all transmitted packets will have port as the source port in the UDP header. Valid values are from 0 to 65535.

RTCPport [in_opt] The default RTCP port number is given by RTP `port` + 1. This parameter makes it possible to use a non-default RTCP port number instead. Valid values are from 1024 to 65535 or GIPS_DEFAULT.

ipaddr [in_opt] To listen on a specific IP interface (if several NICs exists) set `ipaddr` to the desired IP address. If this parameter is set to NULL, each socket will be binded to "0.0.0.0".

multiCastAddr [in_opt] Set this address to a valid multi-cast address if a multi-cast group shall be joined.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

Each new call to `GIPSVE_SetLocalReceiver()` destroys the old sockets (RTP and RTCP) and creates a new pair. Hence, the receiver settings are maintained until `GIPSVE_SetLocalReceiver()` is called again or the channel is deleted.

It is not possible to call this API while listening or sending.

It is possible to break the default port dependency (source port of transmitted packets equals the receiving port set by this API) between receiving and sending sides. See `GIPSVE_SetSendDestination()` for details.

It can sometimes be required to bind the sockets to the local IP address using the optional `ipaddr` parameter. See `GIPSVENetwork::GIPSVE_SetSendGQoS()` as an example.

Example Code

```
// (1) channel 0 will listen on port 12345, RTCP is received on 12345+1 = 12346
base->GIPSVE_SetLocalReceiver(0, 12345);

// (2) same as (1) but using non-default RTCP port (88888)
base->GIPSVE_SetLocalReceiver(0, 12345, 88888);

// (3) same as (1) but specifying what network card to receive on
base->GIPSVE_SetLocalReceiver(0, 12345, GIPS_DEFAULT, "192.168.200.42");

// (4) same as (1) but also joining a multicast group
base->GIPSVE_SetLocalReceiver(0, 12345, GIPS_DEFAULT, NULL, "192.168.200.255");
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_GetLocalReceiver

Retrieves the local receiver port and address for a specified channel number.

Syntax

```
int GIPSVE_GetLocalReceiver(int channel, int& port, int& RTCPport, char* ipaddr,
   unsigned int ipaddrLength);
```

Parameters

channel [in] The channel number ID.

port	[out] A reference to an integer to receive the current RTP port number used for receiving RTP/UDP packets.
RTCPport	[out] A reference to an integer to receive the current RTCP port number used for receiving RTCP packets.
ipaddr	[out] A pointer to a character buffer to receive the IP address of the network card used for receiving. By default, the output string will be empty.
ipaddrLength	[in] Length, in number of characters, of the `ipaddr` character string.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Example Code

```
int port, RTCPport;
char ipaddr[32];

// retrieve local receiver settings for channel 0
base->GIPSVE_GetLocalReceiver(0, port, RTCPport, ipaddr, 32);
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_SetSendDestination

Defines the destination port and address for a specified channel number.

Syntax

```
int GIPSVE_SetSendDestination(int channel, int port, const char* ipaddr, int
  sourcePort = GIPS_DEFAULT, int RTCPport = GIPS_DEFAULT);
```

Parameters

channel	[in] The channel number ID.
port	[in] The RTP/UDP port number to send to. All transmitted packets will contain this port number as the destination port in the UDP header. Valid range is from 1024 to 65535.
ipaddr	[in] Pointer to a zero-terminated character string that contains the IP address to send to.
sourcePort	[in_opt] Modifies the default source port for transmitted RTP/UDP packets. RTCP packets will have source port `sourcePort` + 1. If GIPS_DEFAULT is used, the source port is the same as the receiving port (set by `GIPSVE_SetLocalReceiver()`). Valid values are from 1024 to 65535 or GIPS_DEFAULT.

RTCPport [in_opt] The default RTCP destination port number for transmitted RTPC packets is given by RTP `port` + 1. This optional parameter makes it possible to use a non-default RTCP destination port number instead. Valid values are from 1024 to 65535 or GIPS_DEFAULT.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

`GIPSVE_SetLocalReceiver()` must be called before this function if the destination IP address (`ipaddr`) is a multi-cast address.

If only sending should be enabled, and `GIPSVE_SetLocalReceiver()` has not been called, two different options exists: (1) specify the `sourcePort` in this API to ensure that en extra pair of sending sockets are created directly, or (2) use the default value for `sourcePort`. In the second case, a pair of sending sockets will be created as soon as they are required (at the first packet transmission after calling `GIPSVE_StartSend()`.

If the source port is specified explicitly in this call it is not possible to set the DSCP value explicitly for packets that are sent from this port using `GIPSVE_SetSendTOS()` or `GIPSVE_SetSendGQoS()`.

Example Code

```
// (1) channel 0 will send to 192.168.200.77:55555 (ip:port)
base->GIPSVE_SetSendDestination(0, 55555, "192.168.200.77");

// (2) same as (1) but with source port set to 54321 instead of default
base->GIPSVE_SetSendDestination(0, 55555, "192.168.200.77", 54321);

// (3) same as (1) but RTCP packets will be sent to 11111 instead of 55556
base->GIPSVE_SetSendDestination(0, 55555, "192.168.200.77", GIPS_DEFAULT, 11111);
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_GetSendDestination

Retrieves the destination port and address for a specified channel number.

Syntax

```
int GIPSVE_GetSendDestination(int channel, int& port, char* ipaddr, unsigned int
   ipaddrLength, int& sourcePort, int& RTCPport);
```

Parameters

channel [in] The channel number ID.

port	[out] A reference to an integer to receive the current destination RTP/UDP port number used for sending packets.
ipaddr	[out] A pointer to a character buffer to receive the IP address to which outgoing packets are transmitted.
ipaddrLength	[in] Length, in number of characters, of the `ipaddr` character string.
sourcePort	[out] A reference to an integer to receive the current RTP/UDP source port for transmitted packets.
RTCPport	[out] A reference to an integer to receive the current RTCP port number used for transmitted RTCP packets.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_StartListen

Prepares and initiates the GIPS Voice Engine for listening and reception of incoming RTP/RTCP, packets on the specified channel.

Syntax

```
int GIPSVE_StartListen(int channel);
```

Parameters

channel	[in] The channel number ID.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

This function shall be called also if using external transport, as this function updates RTP packet handling states.

You must call `GIPSVE_SetLocalReceiver()` before this function to ensure that a receiving port is defined, except if using external transport.

Example Code

```
// init
base->GIPSVE_Init();
base->GIPSVE_CreateChannel();
```

```
// start full duplex VoIP call in loopback using PCMU
base->GIPSVE_SetLocalReceiver(0, 12345);
base->GIPSVE_StartListen(0);
base->GIPSVE_StartPlayout(0);
base->GIPSVE_SetSendDestination(0, 12345, "127.0.0.1");
base->GIPSVE_StartSend(0);

// ⇔ full duplex call is now active

// stop call
base->GIPSVE_StopPlayout(0);
base->GIPSVE_StopSend(0);
base->GIPSVE_StopListen(0);

// terminate
base->GIPSVE_DeleteChannel(0);
base->GIPSVE_Terminate();
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_StopListen

Stops receiving incoming RTP/RTCP packets on the specified channel.

Syntax

```
int GIPSVE_StopListen(int channel);
```

Parameters

channel [in] The channel number ID.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

This function shall be called also if using external transport, as this function updates RTP packet handling states.

Example Code

See `GIPSVE_StartListen()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android

VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_StartPlayout

Forwards the packets to the mixer/soundcard for a specific channel.

Syntax

```
int GIPSVE_StartPlayout(int channel);
```

Parameters

channel [in] The channel number ID.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

If you want to mix several channels, `GIPSVE_StartPlayout()` should be called for all channels in the mix. The VoiceEngine automatically mixes all channels that are set to play out incoming data.

Example Code

See `GIPSVE_StartListen()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_StopPlayout

This function stops data from being sent to the mixer/soundcard from the specified channel.

Syntax

```
int GIPSVE_StopPlayout(int channel);
```

Parameters

channel [in] The channel number ID.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

Packets are still received as long as the VoiceEngine is listening to the port.

Example Code
See `GIPSVE_StartListen()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_StartSend
Starts sending packets to an already specified IP address and port number for a specified channel.

Syntax

```
int GIPSVE_StartSend(int channel);
```

Parameters
channel [in] The channel number ID.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks
You must call `GIPSVE_SetSendDestination()` before this function to ensure that a destination address and port is defined.

If only sending should be enabled, and `GIPSVE_SetLocalReceiver()` has not been called, two different options exists: (1) specify the `sourcePort` in `GIPSVE_SetSendDestination()` to ensure that en extra pair of sending sockets are created directly, or (2) use the default value for `sourcePort`. In the second case, a pair of sending sockets will be created as soon as they are required (at the first packet transmission after calling this function).

It is possible to modify the destination by calling `GIPSVE_SetSendDestination()` while transmission is ongoing (on the fly).

Example Code
See `GIPSVE_StartListen()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_StopSend
Stops packets from being sent from a channel.

Syntax

```
int GIPSVE_StopSend(int channel);
```

Parameters

channel [in] The channel number ID.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

This function should be called before `GIPSVE_StopListen()` for a specific channel.

Example Code

See `GIPSVE_StartListen()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_GetVersion

Retrieves the version information for GIPS VoiceEngine and its components.

Syntax

```
GIPSVE_GetVersion(char* version, unsigned int length);
```

Parameters

version [in] Pointer to a character buffer to receive the version string.

length [in] Length, in number of characters, of the `version` character string.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

A buffer size of 1024 characters is sufficient for this call.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_LastError

Retrieves the last VoiceEngine error code.

Syntax

```
int GIPSVE_LastError();
```

Return Values

The return value is positive integer corresponding to the last error that occurred in the VoiceEngine, or -1 if no error has occurred. The positive values are referred to as error codes. See Appendix A: Error Codes for a complete list of error codes.

Remarks

See Error Handling in Chapter 8 for more details on how to interpret the error codes and for recommendations on how to deal with the different categories.

Example Code

```
#include "GIPSVEErrors.h"

if (-1 == base->GIPSVE_StartPlayout(1))
{
    int errCode = base->GIPSVE_LastError();
    if (errCode == VE_CHANNEL_NOT_CREATED)
    {
        // channel 1 is invalid
    }
    else
    {
        // deal with other error codes here
    }
}
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h. All error codes are defined in GIPSVEErrors.h.

GIPSVE_SetConferenceStatus

Toggles the conferencing mode for a specific channel. By adding a channel to a conference, the received audio for this channel will be mixed into the microphone signal that is transmitted to all other channels.

Syntax

```
int GIPSVE_SetConferenceStatus (int channel, bool enable, bool includeCSRCs =
   false, bool includeVoiceLevel = false);
```

Parameters

channel	[in] The channel number ID.
enable	[in] If this flag is `true`, conferencing is enabled. If this flag is `false`, conferencing is disabled.
includeCSRSc	[in_opt] Specifies whether CSRCs should be included in the RTP header (see RFC 3550). Enabling this parameter will make the RTP header correct for voice conferences, but consume more bandwidth.
includeVoiceLevel	[in_opt] Specifies whether voice energy levels of the conference participants shall be added as variable-length header extensions. Each energy level is mapped to values between 0 and 9.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

If CSRC is enabled, the VE mixer inserts a list of the SSRC identifiers of the sources that contributed to the generation of a particular packet into the RTP header of that packet. The mixed result indicates all the talkers whose speech was combined to produce the outgoing packet, allowing the receiver to indicate the current talker, even though all the audio packets contain the same SSRC identifier (that of the VE mixer).

It is only possible to include voice/energy levels in combination with enabled CSRCs.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_PutOnHold

Stops or resumes playout and transmission on a temporary basis.

Syntax

```
int GIPSVE_PutOnHold(int channel, bool enable, GIPS_OnHoldModes mode =
  HOLD_SEND_AND_PLAY);
```

Parameters

channel	[in] The channel number ID.
enable	[in] If this parameter is `true`, the call is put on hold (stops playout and transmission for the default mode parameter). If the parameter is `false`, the call is resumed again.
mode	[in_opt] A `GIPS_OnHoldModes` enumerator that specifies which direction should be affected by the on-hold operation.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

The default `mode` is `HOLD_SEND_AND_PLAY`.

If `mode` is set to `HOLD_SEND_ONLY`, the recorded audio will not be encoded or transmitted. However, scheduled RTCP packets and RTP keepalive packets are still transmitted.

If `mode` is set to `HOLD_PLAY_ONLY`, audio is still played out, but the original output stream is replaced by an all-zero signal.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_SetNetEQPlayoutMode

Defines the NetEQ playout mode for a specified channel number.

Syntax

```
int GIPSVE_SetNetEQPlayoutMode(int channel, GIPS_NetEQModes mode);
```

Parameters

channel [in] The channel number ID.

mode [in] A `GIPS_NetEQModes` enumerator that sets what type of playout mode to enable.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

In fax playout mode, the jitter buffer is optimized to enforce a constant delay instead of maintaining a low delay. The rationale is that delay changes can be detrimental to fax transmissions (or another sensitive signal such as TTY/TDD), while the delay is not as important as it is in conversation. The fax mode should only be used for fax and modem transmissions, since the perceived quality is worse in this mode when used for voice communication.

It is recommended to use the fax mode for as short time as possible, since the delay will be higher.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone, Android
VE configuration	Standard

Header	Declared in GIPSVEBase.h

GIPSVE_GetNetEQPlayoutMode

Retrieves the current NetEQ playout mode for a specific channel.

Syntax

```
int GIPSVE_GetNetEQPlayoutMode(int channel, GIPS_NetEQModes& mode);
```

Parameters

channel [in] The channel number ID.

mode [out] A `GIPS_NetEQModes` enumerator which specifies the current playout mode on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_SetNetEQBGNMode

Defines the NetEQ background noise mode for a specified channel number.

Syntax

```
int GIPSVE_SetNetEQBGNMode(int channel, GIPS_NetEQBGNModes mode);
```

Parameters

channel [in] The channel number ID.

mode [in] A `GIPS_NetEQBGNModes` enumerator that sets what type of BGN mode to enable.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

The default mode (used when this function has not been called) is `GIPS_BGN_ON`.

If the incoming RTP stream stops abnormally (i.e., not during VAD/DTX silence periods) NetEQ will at first try to extrapolate the latest speech signal to produce an output signal while waiting for the stream to resume. If the interruption last for a longer time, the synthetic speech extrapolation can be replaced with a background noise, generated internally from parameters estimated previously from the incoming signal.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVE_GetNetEQBGNMode

Retrieves the current NetEQ BGN mode for a specific channel.

Syntax

```
int GIPSVE_GetNetEQBGNMode(int channel, GIPS_NetEQBGNModes& mode);
```

Parameters

channel [in] The channel number ID.

mode [out] A `GIPS_NetEQBGNModes` enumerator which specifies the current BGN mode on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVEBase.h

GIPSVECodec

The `GIPSVECodec` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Support of non-default codecs (e.g. iLBC, iSAC, G.729 etc.).
- Voice Activity Detection (VAD) on a per channel basis.
- Possibility to specify what codec a received payload type shall be mapped to.
- Additional AMR encoder and decoder settings.
- Bandwidth Extension (BWE) functionality.

NOTE: The `GIPSVECodec`:: prefix is excluded for most API names throughout this chapter.

Enumerator GIPS_AMRmodes

This enumerator is used to specify the RTP payload format to be used for Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) encoded speech signals. It is utilized by the `GIPSVE_SetAMREncFormat()` and `GIPSVE_SetAMRDecFormat()` APIs.

Syntax

```
enum GIPS_AMRmodes
{
    AMR_RFC3267_BWEFFICIENT = 0,
    AMR_RFC3267_OCTETALIGNED,
    AMR_RFC3267_FILESTORAGE
};
```

Enumerators

AMR_RFC3267_BWEFFICIENT	Bandwidth efficient payload format.
AMR_RFC3267_OCTETALIGNED	Octet aligned payload format.
AMR_RFC3267_FILESTORAGE	File storage payload format.

Remarks

See RFC 3267 for additional details.

Enumerator GIPS_PayloadFrequencies

This enumerator is used to set the frequency at which comfort noise will be generated. It is used by the `GIPSVE_SetSendCNPayloadType()` method.

Syntax

```
enum GIPS_PayloadFrequencies
{
    FREQ_8000_HZ = 8000,
    FREQ_16000_HZ = 16000
};
```

Enumerators

FREQ_8000_HZ	Sample rate of 8 000 samples per second [Hz].
FREQ_16000_HZ	Sample rate of 16 000 samples per second [Hz].

Enumerator GIPS_VADmodes

This enumerator is used to set the degree of bandwidth reduction for GIPS Voice Activity Detection (VAD). It is utilized by the `GIPSVE_SetVADStatus()` API.

Syntax

```
enum GIPS_VADmodes
{
    VAD_CONVENTIONAL = 0,
    VAD_AGGRESSIVE_LOW,
    VAD_AGGRESSIVE_MID,
    VAD_AGGRESSIVE_HIGH
};
```

Enumerators

VAD_CONVENTIONAL	The lowest bandwidth reduction.
VAD_AGGRESSIVE_LOW	A bandwidth reduction higher than VAD_CONVENTIONAL but lower than VAD_AGGRESSIVE_MID.
VAD_AGGRESSIVE_MID	A bandwidth reduction higher than VAD_AGGRESSIVE_LOW but lower than VAD_AGGRESSIVE_HIGH.
VAD_AGGRESSIVE_HIGH	The highest bandwidth reduction.

GetInterface

Retrieves a pointer to the GIPSVECodec sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVECodec* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVECodec` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()`method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android

VE configuration	Standard
Header	Declared in GIPSVECodec.h

Release

Releases the GIPSVECodec sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_NumOfCodecs

Retrieves the number of supported codecs in this particular build of GIPS VoiceEngine.

Syntax

```
int GIPSVE_NumOfCodecs();
```

Return Values

The return value is always a positive integer which corresponds to the number of supported codecs.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetCodec

Retrieves the codec information for a specified list index.

Syntax

```
int GIPSVE_GetCodec(int index, GIPS_CodecInst& codec);
```

Parameters

index [in] The requested codec in the internal prioritized codec list (0=highest priority).

codec [out] A `GIPS_CodecInst` structure which is filled in with codec information on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Use `GIPSVE_NumOfCodecs()` to get the length of the list.

Example Code

```
// acquire sub-API (assuming GIPSVoiceEngine object exists)
GIPSVECodec* codec = GIPSVECodec::GetInterface(ve);

// list all supported codecs
for (int = 0; i < codec->GIPSVE_NumOfCodecs(); i++)
{
    GIPS_CodecInst cinst;
    codec->GIPSVE_GetCodec(i, cinst);
    DISPLAY_CODEC_INFO(i, cinst);
}
// release sub-API
codec->Release();
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetSendCodec

This function sets the codec for the channel to be used for sending. The codec information is part of the input arguments since the payload type and packet size can vary.

Syntax

```
int GIPSVE_SetSendCodec(int channel, const GIPS_CodecInst& codec);
```

Parameters

channel [in] The channel ID number.

codec [in] The `GIPS_CodecInst` structure holding the codec information.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Payload name and sampling frequency (MIME information) must be the same as those types supported by the GIPS VoiceEngine.

Example Code

```
// This example assumes that a GIPSVECodec sub-API pointer exists.

GIPS_CodecInst cinst;

// define GIPS iSAC codec parameters
strcpy(cinst.plname, "ISAC");
cinst.plfreq = 16000;  // iSAC wideband mode
cinst.pltype = 103;    // default dynamic payload type
cinst.pacsize = 480;   // use 30ms packet size
cinst.channels = 1;
cinst.rate = -1;       // channel-adaptive mode

// activate iSAC for channel 0
codec->GIPSVE_SetSendCodec(0, cinst);
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetSendCodec

This function retrieves the codec parameters for the sending codec on a specified channel.

Syntax

```
int GIPSVE_GetSendCodec(int channel, GIPS_CodecInst& codec);
```

Parameters

channel [in] The channel ID number.

codec [out] A `GIPS_CodecInst` structure which is filled in with codec information on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetRecCodec

This function returns the currently received codec for a specific channel.

Syntax

```
int GIPSVE_GetRecCodec(int channel, GIPS_CodecInst& codec);
```

Parameters

channel [in] The channel ID number.

codec [out] A `GIPS_CodecInst` structure which is filled in with codec information on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetSendCodecAuto

This call enables automatic switching between the iSAC and Speex codecs. A call is started in iSAC mode and if the iSAC bit rate becomes too low, this function switches the codec to Speex at 8000 bit/s.

Syntax

```
int GIPSVE_SetSendCodecAuto(int channel, bool enable, int isacPT, int speexPT);
```

Parameters

channel [in] The channel ID number.

enable [in] If this parameter is `true`, automatic codec switching mode is enabled. If the parameter is `false`, automatic codec switching mode is disabled.

isacPT [in] Payload type for the iSAC codec.

speexPT [in] Payload type for the Speex codec.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetAMREncFormat
This call is specific to the AMR encoder. It sets the packet format for the AMR narrowband encoder.

Syntax

```
int GIPSVE_SetAMREncFormat(int channel, GIPS_AMRmodes mode =
   AMR_RFC3267_BWEFFICIENT);
```

Parameters
channel [in] The channel ID number.

mode [in_opt] A `GIPS_AMRmodes` enumerator that sets what type of AMR encoder format to use.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks
The formats are further described in RFC 3267.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetAMRDecFormat
This call is specific to the AMR decoder. It sets the packet format for the AMR narrowband decoder.

Syntax

```
int GIPSVE_SetAMRDecFormat(int channel, GIPS_AMRmodes mode =
   AMR_RFC3267_BWEFFICIENT);
```

Parameters

channel	[in] The channel ID number.
mode	[in_opt] A `GIPS_AMRmodes` enumerator that sets what type of AMR decoder format to use.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The formats are further described in RFC 3267.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetISACInitTargetRate

The API sets the initial values of target rate and frame size for iSAC for a specified channel. This API is only valid if iSAC is setup to run in channel-adaptive mode (see example at page 61).

Syntax

```
int GIPSVE_SetISACInitTargetRate(int channel, int rateBps, bool useFixedFrameSize
   = false);
```

Parameters

channel	[in] The channel ID number.
rateBps	[in] Initial iSAC target rate in bits/second. Valid range is 10000-56000 bps. If `rateBps` is set to 0, a default initial target rate of 20000 bps in wideband mode and 56000 bps in super-wideband mode will be used.
useFixedFrameSize	[in_opt] If this parameter is `true`, the frame size will be fixed at the size previously set by `GIPSVE_SetSendCodec`, while the rate varies. If the parameter is `false`, iSAC (in wideband mode) can automatically change the frame size back and forth between 30ms and 60ms.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This API will only have an effect if the sending encoder is set to channel-adaptive mode of iSAC. See `GIPSVE_SetSendCodec()` for more details and the example below.

Example Code

```
// This example assumes that a GIPSVECodec sub-API pointer exists.

GIPS_CodecInst cinst;

// define GIPS iSAC codec parameters
strcpy(cinst.plname, "ISAC");
cinst.plfreq = 16000;    // iSAC wideband mode
cinst.pltype = 103;      // default dynamic payload type
cinst.pacsize = 480;     // use 30ms packet size
cinst.rate = -1;         // use channel-adaptive rate
cinst.channels = 1;      // NA

// set 30ms adaptive rate iSAC for channel 0 (default initial target rate is 20000 bps)
codec->GIPSVE_SetSendCodec(0, cinst);
// (1) override default initial target rate and keep variable frame size, or
codec->GIPSVE_SetISACInitTargetRate(0, 32000, false);

// (2) override default initial target rate and use fixed (=30ms) frame size, or
codec->GIPSVE_SetISACInitTargetRate(0, 32000, true);

// (3) restore default adaptive iSAC mode
codec->GIPSVE_SetISACInitTargetRate(0, 0);
```

Requirements

Supported platforms	Windows, MAC OS X, Linux, Symbian
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetISACMaxRate

Sets the maximum allowed iSAC rate which the codec may not exceed for a single packet for the specified channel. The maximum rate is defined as payload size per frame size in bits per second.

Syntax

```
int GIPSVE_SetISACMaxRate(int channel, int rateBps);
```

Parameters

channel [in] The channel ID number.

rateBps [in] The maximum rate is in bits/second. Valid values are between 32000 and 53400 in wideband mode, and between 32000 and 160000 in super-wideband mode, in steps of 100. Smaller resolution than that will not have any effect. Set to 53400/160000 to restore default limitation.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This API will only have an effect if the sending encoder is set to iSAC.

This function limits each packet (not the average) to the specified rate. Depending on if 30 ms or 60 ms packets are transmitted, the maximum payload size will differ with a factor 2.

It is possible to call this API for both channel-adaptive and non-adaptive iSAC modes.

This function must be called before `GIPSVE_StartSend()` for a specified channel.

This function can be used in conjunction with `SetISACMaxPayloadSize()`. See remarks in that API description for details.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetISACMaxPayloadSize

Sets the maximum allowed iSAC payload size for a specified channel. The maximum value is set independently of the frame size, i.e. 30 ms and 60 ms packets have the same limit.

Syntax

```
int GIPSVE_SetISACMaxPayloadSize(int channel, int sizeBytes);
```

Parameters

channel [in] The channel ID number.

sizeBytes [in] Maximum size of payload in bytes. Valid range is 100-400 bytes in wideband mode, and 100-600 bytes in super-wideband mode. Setting `sizeBytes` to 400, restores the default limitation.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This API will only have an effect if the sending encoder is set to iSAC.

It is possible to call this API for both channel-adaptive and non-adaptive iSAC modes.

This function must be called before `GIPSVE_StartSend()` for a specified channel.

This function can be used in conjunction with `GIPSVE_SetISACMaxRate()`. For each packet encoded, the maximum payload size will in effect be limited by the strongest limitation of the two. Since

GIPSVE_SetISACMaxRate() will limit the payload size for a single packet differently for 30 ms and 60 ms packets, it is possible that the settings from one of the functions will be a stronger limit for one packet size, and the settings from the other function will be a stronger limit for the other packet size.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetRecPayloadType

This function is used to set the dynamic payload type number for a particular codec or to disable (ignore) a codec for receiving. For instance, when receiving an invite from a SIP-based client, this function can be used to change the dynamic payload type number to match that in the INVITE SDP-message. The utilized parameters in the codec structure are `plname`, `plfreq`, `pltype` and `channels`.

Syntax

```
int GIPSVE_SetRecPayloadType(int channel, const GIPS_CodecInst& codec);
```

Parameters

channel [in] The channel ID number.

codec [in] The `GIPS_CodecInst` structure holding the codec information including the modified payload type number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Set `pltype` to -1 to disable (ignore) the codec. To re-enable the codec, call the function again with the desired pltype. `GIPSVE_GetCodec()` can be used to get the default settings for a codec.

This function can only be called when not listening or playing.

After `StopListen`() has been called, all available codecs will be enabled with default payload types. The desired changes must then be done again.

It is possible to receive stereo-audio packetized according to RFC 3551 (http://www.ietf.org/rfc/rfc3551.txt). To do so, ensure that the `channels` parameter is set to 2 (=stereo). It will ensure that all incoming RTP packets, for the specified payload type, will be decoded and played out in stereo. The exact decoding scheme is dependent on the codec name and follows Table 1 in Section 4.2 of RTC 3551. See the example below for more details.

NOTE: This stereo playout is only supported on Windows, Mac OS X and LINUX ALSA.

Example Code

```
// This example exemplifies how to set up the receiver for stereo playout.
```

```
GIPS_CodecInst cinst;

// define stereo 48kbs G.722.1C @ 32kHz
strcpy(cinst.plname, "G7221");
cinst.plfreq = 32000;    // G.722.1C is a super-wideband codec
cinst.pltype = 125;      // specify dynamic payload type
cinst.pacsize = 640;     // 20ms packet size
cinst.rate = 48000;      // 48kbps mode
cinst.channels = 2;      // receive as stereo codec

// ensure that incoming G.722.1C RTP packets at PT=125 is decoded as a stereo codec
codec-> GIPSVE_SetRecPayloadType(0, cinst);
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetRecPayloadType

This function retrieves the actual payload type that is set for receiving a codec on a channel. The value it retrieves will either be the default payload type, or a value earlier set with `GIPSVE_SetRecPayloadType()`.

Syntax

```
int GIPSVE_GetRecPayloadType(int channel, GIPS_CodecInst& codec);
```

Parameters

channel [in] The channel ID number.

codec [in/out] The `GIPS_CodecInst` structure holding the codec information. This function will only look at the `codec.plname`, `codec.plfreq`, and `codec.channels` parameters, and write the result to `codec.pltype`.

Return Values

The return value is `0` if the function succeeds. The actual result will be written to codec.pltype. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

To retrieve the payload type, this function will use the codec name, the codec frequency (since some codec might use two different frequencies with different payload types) and also the number of channels (for stereo or mono).

Note that, this function both writes to and reads from the codec parameter, hence the [in/out] notation above.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetSendCNPayloadType

Sets the payload type for the sending of SID-frames with background noise estimation during silence periods detected by the VAD (Voice Activity Detection).

Syntax

```
int GIPSVE_SetSendCNPayloadType(int channel, int type, GIPS_PayloadFrequencies
   frequency = FREQ_8000_HZ);
```

Parameters

channel	[in] The channel ID number.
type	[in] The payload type number for comfort noise.
frequency	[in_opt] A `GIPS_PayloadFrequencies` enumerator that sets the comfort noise frequency.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetVADStatus

This function enables or disables the VAD/DTX (silence suppression) functionality for a specified channel.

Syntax

```
int GIPSVE_SetVADStatus(int channel, bool enable, GIPS_VADmodes mode =
   VAD_CONVENTIONAL, bool disableDTX = false);
```

Parameters

channel	[in] The channel ID number.
enable	[in] If this parameter is `true`, VAD/DTX is enabled. If the parameter is `false`, VAD/DTX is disabled.

mode [in_opt] A `GIPS_VADmodes` enumerator that sets the degree of bandwidth reduction for GIPS VAD.

disableDTX [in_opt] If enabled (and the VAD/DTX is enabled), the DTX will be disabled while the VAD is enabled. The VAD/DTX will then detect silent frames but let all sound through unaffected.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The voice activity detection (VAD) feature is used to determine if frames of audio contain speech or silence. When coupled with discontinuous transmission (DTX), the VoiceEngine will send much smaller comfort noise (CN) packets during silence periods, thereby decreasing the transmission bitrate.

The `disableDTX` parameter is useful for getting information on the VAD decisions without affecting the sound. The VAD decision can be extracted with `GIPSVEVQE::GIPSVE_VoiceActivityIndicator()`.

`mode` and `disableDTX` are ignored if VAD/DTX is disabled.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetVADStatus

Retrieves the current VAD/DTX status and mode settings for a specified channel.

Syntax

```
int GIPSVE_GetVADStatus(int channel, bool& enabled, GIPS_VADmodes& mode, bool&
   disabledDTX);
```

Parameters

channel [in] The channel ID number.

enabled [out] A binary reference output which is set to `true` if VAD/DTX is enabled and `false` otherwise.

mode [out] A `GIPS_VADmodes` enumerator which will contain the current VAD/DTX mode on return.

disableDTX [out] A binary reference output which is set to `true` if DTX is disabled and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetBandwidthExtensionStatus

This function upsamples and artificially extends the bandwidth of a wideband (16kHz sampling rate) signal. The described functionality is called Bandwidth Extension (BWE).

Syntax

```
int GIPSVE_SetBandwidthExtensionStatus(bool enable);
```

Parameters

enable [in] If this parameter is `true`, BWE is enabled. If the parameter is `false`, BWE is disabled.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Bandwidth extension is performed on the mixed output signal before the signal is sent to the playout side of the soundcard.

The BWE operation will only have an effect on input signals produced by wideband codecs (e.g. iPCM-wb, iSAC, G.722 etc.), all using a sampling rate of 16kHz.

Bandwidth extension can be ignored internally even if BWE has been successfully enabled. It will be ignored for the following conditions:

- If the output sampling rate is not set to 48 kHz. It is e.g. possible to select another output sampling rate on Windows Vista. For all other platforms, 48kHz is used by default.
- If any stereo function is enabled.

A warning message will be added to the trace file if any of the conditions above are detected while BWE is enabled.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetBandwidthExtensionStatus

Retrieves the current BWE status.

Syntax

```
int GIPSVE_GetBandwidthExtensionStatus(bool& enabled);
```

Parameters

enabled [out] A binary reference output which is set to `true` if BWE is enabled and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_ConfigureChannel

This function configures RTP packet size and DTX properties for a channel using the fmtp string from the SDP message as input. This call can parse the following parameters:

- ptime
- cng
- annexb (for G.729 DTX)
- ebw (for Speex)

Syntax

```
int GIPSVE_ConfigureChannel(int channel, const char* mimeKeyValues);
```

Parameters

channel [in] The channel ID number.

mimeKeyValues [in] Pointer to string containing the fmtp parameters. Example: "ptime=30;cng=on".

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_GetChannelMIMEParameters

This function retrieves the RTP packet size and DTX properties for a specified channel.

Syntax

```
int GIPSVE_GetChannelMIMEParameters(int channel, char* buf, unsigned int length);
```

Parameters

channel [in] The channel number ID.

buf [out] A pointer to a character buffer to receive the channel MIME parameters.

length [in] Length, in number of characters, of the `buf` character string.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

See `GIPSVE_ConfigureChannel()` for example output.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVENetwork

The `GIPSVENetwork` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- External protocol support.
- Extended port and address APIs.
- Port and address filters.
- Windows GQoS functions.
- Packet timeout notification.
- Dead-or-Alive connection observations.
- Transmission of raw RTP/RTCP packets into existing channels.

NOTE: The `GIPSVENetwork`:: prefix is excluded for most API names throughout this chapter.

Class GIPSVEConnectionObserver

This class declares an abstract interface for a user definable observer mechanism. It is up to the VoiceEngine user to implement a derived class which implements the observer class. The observer is installed and activated by the `GIPSVE_SetDeadOrAliveObserver()` and `GIPSVE_SetPeriodicDeadOrAliveStatus()` APIs respectively.

NOTE: Always ensure that the callback functions are kept as short as possible to ensure that additional callbacks are not delayed.

```
class GIPSVEConnectionObserver
{
public:
  virtual void OnPeriodicDeadOrAlive(int channel, bool alive) = 0;
};
```

GIPSVEConnectionObserver::OnPeriodicDeadOrAlive

This method will be called peridically and deliver dead-or-alive decisions for a specified channel when the observer interface has been installed and activated.

Syntax

```
void OnPeriodicDeadOrAlive(int channel, bool alive);
```

Parameters

channel	[out] The channel ID number.
alive	[out] The binary dead-or-alive decision sent to the observer, where `true` means that the channel is 'Alive' and `false` means that the channel is 'Dead'.

Remarks

Each binary dead-or-alive decision is based on a mix of variables for each channel, e.g., time since last valid RTP packet was received, comfort noise state, and some additional internal factors.

GetInterface

Retrieves a pointer to the `GIPSVENetwork` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVENetwork* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVENetwork` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

Release

Releases the `GIPSVENetwork` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVECodec.h

GIPSVE_SetExternalTransport

This function call enables or disables a user-defined external transport protocol for a specified channel.

Syntax

```
int GIPSVE_SetExternalTransport(int channel, bool enable, GIPS_transport*
   transport);
```

Parameters

channel [in] The channel ID number.

enable [in] If this parameter is `true`, external transport is enabled. If the parameter is `false`, external transport is disabled.

transport [in] Pointer to an implemented `GIPS_transport` class .

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

It is up to the VoiceEngine user to implement a class which overrides `GIPS_transport::SendPacket()` and `GIPS_transport::SendRTCPPacket()`. These two methods will, upon activation, be called by the VoiceEngine for each block of data that is recorded, encoded and packetized into RTP or RTCP packets.

The `transport` pointer is ignored if `enable` is set to false.

See the `GIPS_transport` description for more details.

`GIPSVE_StartListen()` and `GIPSVE_StopListen()` shall always be called also when using external transport, since they update RTP packet handling states.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_ReceivedRTPPacket

The packets received from the network should be passed to this function when external transport is enabled. Note that the data including the RTP-header must also be given to the VoiceEngine.

Syntax

```
int GIPSVE_ReceivedRTPPacket(int channel, const void* data, unsigned int length);
```

Parameters

channel [in] The channel ID number.

data [in] Pointer to data buffer which contains the received RTP packet.

len [in] Length, in number of bytes, of the `data` buffer.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This function call is only valid if external transport is enabled by `GIPSVE_SetExternalTransport()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_ReceivedRTCPPacket

The packets received from the network should be passed to this function when external transport is enabled. Note that the data including the RTCP-header must also be given to the VoiceEngine.

Syntax

```
int GIPSVE_ReceivedRTCPPacket(int channel, const void* data, unsigned int
   length);
```

Parameters

channel [in] The channel ID number.

data [in] Pointer to data buffer which contains the received RTCP packet.

len [in] Length, in number of bytes, of the `data` buffer.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This function call is only valid if external transport is enabled by `GIPSVE_SetExternalTransport()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_GetSourceInfo

This function retrieves the source ports and IP address of incoming packets on a specific channel.

Syntax

```
int GIPSVE_GetSourceInfo(int channel, int& rtpPort, int& rtcpPort, char* ipaddr,
   unsigned int ipaddrLength);
```

Parameters

channel [in] The channel ID number.

rtpPort [out] An integer reference where the source RTP port will be placed.

rtcpPort [out] An integer reference where the source RTCP port will be placed.

ipaddr [out] A pointer to an array to which the source IP address will be copied as a null-terminated string.

ipaddrLength [in] The size of the array pointed to by `ipaddr` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_GetLocalIP

This function copies the local (host) IP address, in string format, to the provided buffer.

Syntax

```
int GIPSVE_GetLocalIP(char* ipaddr, unsigned int ipaddrLength, bool ipv6 =
   false);
```

Parameters

ipaddr [out] A pointer to an array to which the local IP address will be copied as a null-terminated string.

ipaddrLength [in] The size of the array pointed to by `ipaddr` in bytes.

ipv6 [in] If this parameter is set to `true` the IPv6 address will be returned.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

A buffer size of 128 characters is sufficient for this call. The IPv4 address will be returned by default.

The IPv6 address cannot be returned for Windows CE/Mobile and Android.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_EnableIPv6

This function enables IPv6 for a specified channel.

Syntax

```
int GIPSVE_EnableIPv6(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

IP version 4 is used by default for all channels if no modification has been done using this function.

This function must be called before `GIPSVE_SetLocalReceiver()` and `GIPSVE_SetSendDestination()`.

It is not possible to modify the IP version while listening or sending is active.

If IPv6 has been enabled using this API, the only way to restore the default IPv4 protocol is to delete the channel and then create it again.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_IPv6IsEnabled

This function returns `true` if IPv6 is enabled and `false` if IPv6 is disabled for a specified channel.

Syntax

```
bool GIPSVE_IPv6IsEnabled(int channel);
```

Parameters

channel [out] The channel ID number.

Return Values

The return value is `true` if IPv6 is enabled and `false` if IPv6 is disabled (corresponding to a state where IPv4 is used).

Remarks

This function should be called after `GIPSVE_EnableIPv6()` to verify that IPv6 has been enabled correctly.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_SetSourceFilter

This function enables a port and IP address filter for incoming packets on a specific channel.

Syntax

```
int GIPSVE_SetSourceFilter(int channel, int rtpPort, int rtcpPort = 0,
   const char* ipaddr = NULL);
```

Parameters

channel	[in] The channel ID number.
rtpPort	[in] RTP/UDP filter port number. Only packets originating from this source port will be accepted.
rtcpPort	[in] RTCP/UDP filter port number. Only packets originating from this source port will be accepted.
ipaddr	[in] A pointer to an array containing an IP address as a null-terminated string. Only packets originating from this source IP will be accepted.

Remarks

The incoming packet must fulfill both the port and the IP address filter to be accepted. See the example code below for more details.

To disable the port filter, set `port` to `0`.

To disable the address filter, set the `ipaddr` `NULL`.

Example Code

```
// This example assumes that a full duplex VoIP session is active.

int sourceRtpPort(-1);
int sourceRtcpPort(-1);
char sourceIP[32] = {0};

// acquire sub-API (assuming GIPSVoiceEngine object exists)
GIPSVENetwork* netw = GIPSVENetwork::GetInterface(ve);
```

```
// retrieve source port and IP address of incoming packets for channel 0
netw->GIPSVE_GetSourceInfo(0, sourceRtpPort, sourceRtcpPort, sourceIP, 32);

// set filter which allows the incoming stream to pass
netw->GIPSVE_SetSourceFilter(0, sourceRtpPort, sourceRtcpPort, sourceIP);

// modify filter port => incoming stream is now blocked
netw->GIPSVE_SetSourceFilter(0, sourceRtpPort+10, sourceRtcpPort+10, sourceIP);

// disable port filter => incoming stream is now received again
netw->GIPSVE_SetSourceFilter(0, 0, 0, sourceIP);

// modify filter IP address => incoming stream is now blocked
netw->GIPSVE_SetSourceFilter(0, sourceRtpPort, sourceRtcpPort, "10.10.10.10");

// disable IP filter => incoming stream is now received again
netw->GIPSVE_SetSourceFilter(0, sourceRtpPort, sourceRtcpPort, NULL);

// disable all filters
netw->GIPSVE_SetSourceFilter(0, 0, 0, NULL);

// release sub-API
netw->Release();
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_GetSourceFilter

Retrieves the current port and IP-address filter for a specified channel.

Syntax

```
int GIPSVE_GetSourceFilter(int channel, int& rtpPort, int& rtcpPort, char*
   ipaddr, unsigned int ipaddrLength);
```

Parameters

channel [in] The channel ID number.

rtpPort [out] An integer reference where the RTP/UDP filter port will be placed on return.

rtcpPort [out] An integer reference where the RTCP/UDP filter port will be placed on return.

ipaddr [out] A pointer to an array to which the IP address filter will be copied as a null-terminated string.

ipaddrLength [in] The size of the array pointed to by `ipaddr` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If no IP address filter has been set, `ipaddr` will be empty upon return.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_SetSendTOS

This function sets the six-bit Differentiated Services Code Point (DSCP) in the IP header of the outgoing stream for a specific channel.

Syntax

```
int GIPSVE_SetSendTOS(int channel, int DSCP, bool useSetSockopt = false);
```

Parameters

channel	[in] The channel ID number.
DSCP	[in] The six-bit DSCP value. Valid range is 0-63. As defined in RFC 2472, the DSCP value is the high-order 6 bits of the IP version 4 (IPv4) TOS field and the IP version 6 (IPv6) Traffic Class field.
useSetSockopt	[in_opt] If this parameter is `true`, the Windows Socket (Winsock) function `setsockopt()` is used internally. If the parameter is false, traffic control APIs are utilized instead.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

It is recommended to use `GIPSVE_SetSendGQoS`() on Windows instead of this function if possible.

This function must always be called after `GIPSVEBase::GIPSVE_SetLocalReceiver()`since it requires that sockets already exists.

The `useSetSockopt` parameter is ignored on Linux and on MAC OS X. It is always interpreted as `true` internally, i.e., using the `setsockopt()` API is the only option on Linux and MAC OS X.

By default (on Windows 2000/XP/2003) you must first specify a receiving IP address by calling `GIPSVEBase::GIPSVE_SetLocalReceiver()`. The NIC for a socket is found by IP address when dealing with Windows Traffic Control. Binding to the local IP address is not required if `useSetSockopt` is set to `true`. Use `GIPSVENetwork::GIPSVE_GetLocalIP()` to retrieve the local IP address.

According to http://support.microsoft.com/kb/248611: *Microsoft Windows 2000, Microsoft Windows XP, and Microsoft Windows Server 2003 do not support the marking of Internet Protocol (IP) Type of Service (ToS) bits with the setsockopt() function*.

Setting the DSCP value on Windows requires that the executable runs with Administrator privileges. The DSCP value will not be modified unless this condition is fulfilled.

It is possible to modify the DSCP value “on the fly”, i.e., while sending is active. However, it is recommended to consider this as a permanent setting for each RTP session.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_GetSendTOS

Retrieves the six-bit Differentiated Services Code Point (DSCP) in the IP header of the outgoing stream for a specific channel.

Syntax

```
GIPSVE_GetSendTOS(int channel, int& DSCP, bool& useSetSockopt);
```

Parameters

channel [in] The channel ID number.

DSCP [out] An integer reference where the six-bit DSCP will be placed on return.

useSetSockopt [out] A binary reference output which is set to `true` if the Windows Socket (Winsock) function `setsockopt()` is used internally. It is set to `false` if traffic control APIs are utilized instead.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_SetSendGQoS

This function sets the Generic Quality of Service (GQoS) service level. The Windows operating system then maps to a Differentiated Services Code Point (DSCP) and to an 802.1p setting.

Syntax

```
int GIPSVE_SetSendGQOS(int channel, bool enable, int serviceType, int
   overrideDSCP = 0);
```

Parameters

channel	[in] The channel ID number.
enable	[in] If this parameter is `true`, GQoS is enabled. If the parameter is `false`, GQoS is disabled.
serviceType	[in] The GQoS service type. The Windows operating system then maps to a Diffserv codepoint (DSCP) and to an 802.1p setting. See the GQoS table below for more details.
overrideDSCP	[in_opt] Specifying this parameter overrides the DSCP value as mapped from the `serviceType` value, and the traffic control APIs will be used internally. If set to 0, the QoS APIs and normal mapping from `serviceType` will be used. See the Remarks section for more details.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Setting the GQoS service level on Windows requires that the executable runs with Administrator privileges. The GQoS service level will not be modified unless this condition is fulfilled.

This function must be called after both `GIPSVEBase::GIPSVE_SetLocalReceiver()` and `GIPSVEBase::GIPSVE_SetSendDestination()` to have any effect.

On Windows 2000/XP/2003, you must specify a local IP when calling `GIPSVEBase::GIPSVE_SetLocalReceiver()` if `overrideDSCP` is specified (i.e. > 0). The NIC for a socket is found by IP address when dealing with Windows Traffic Control. `GIPSVENetwork::GIPSVE_GetLocalIP()` can be utilized to retrieve the local IP address.

The Windows GQoS API is used to modify both the DSCP and 802.1p marker bits. This function does this by setting a GQoS service level. The Windows operating system maps this to corresponding DSCP and 802.1p settings. The following table lists the supported default GQoS values. See http://technet.microsoft.com/en-us/library/cc787218(WS.10).aspx for more details.

Service Type Name	serviceType value (defined in qos.h)	DSCP	802.1p
Guaranteed Service	SERVICETYPE_GUARANTEED	0x28 (class selector 5)	5
Controlled Load	SERVICETYPE_CONTROLLEDLOAD	0x18 (3)	3
Qualitative	SERVICETYPE_QUALITATIVE	0x0 (0)	0
Best Effort	SERVICETYPE_BESTEFFORT	0x0 (0)	0

Using overrideDSCP will utilize the traffic control APIs, similar to when SetSendTOS (without setsockopt) is called. The difference is that SetSendGQoS will set up an internally specified flow specification, including serviceType and other parameters. SetSendTOS sets up a flow specification containing default values and unspecified parameters. Refer to the MSDN library for details.

In order to change the DSCP value when using overrideDSCP, GQoS must first be disabled and then enabled again with the new value.

It is possible to modify the DSCP value “on the fly”, i.e., while sending is active. However, it is recommended to consider this as a permanent setting for each RTP session.

Requirements

Supported platforms	Windows
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_GetSendGQoS

This function retrieves the currently set GQoS service level for a specific channel.

Syntax

```
int GIPSVE_GetSendGQOS(int channel, bool& enabled, int& serviceType, int&
   overrideDSCP);
```

Parameters

channel [in] The channel ID number.

enabled [out] The current GQoS state. If enabled is set to `true`, GQoS is enabled. If enabled is set to `false`, GQoS is disabled.

serviceType [out] The GQoS service level is placed here on return.

overrideDSCP [out] The non-default DSCP value (overrides the default mapping according to the GQoS table above) is placed here on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Requirements

Supported platforms	Windows
VE configuration	Standard
Header	Declared in GIPSVENetwork.h

GIPSVE_SetPacketTimeoutNotification

This function enables or disables warnings that report if packets have not been received in `timeoutSeconds` seconds for a specific channel.

Syntax

```
int GIPSVE_SetPacketTimeoutNotification(int channel, bool enable, int
   timeoutSeconds);
```

Parameters

channel [in] The channel ID number.

enable [in] If set to `true`, packet-timeout notification is enabled. If set to `false`, packet-timeout notification is disabled.

timeoutSeconds [in] Time-out time in seconds. A notification will be triggered if a packet has not arrived within this time.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The `timeoutSeconds` value should be within the interval 1 < `timeoutSeconds` < 150 seconds.

When packet-timeout notification is enabled, the user gets a callback if no packet has been received within the specified time (1-150 seconds), and if `GIPSVE_Base::GIPSVE_StartListen()` has been called.

To receive callback messages, the `GIPSVoiceEngineObserver::CallbackOnError()` method must be implemented and the observer must be activated using `GIPSVEBase::GIPSVE_SetObserver()`.

The callback message `errCode` at packet timeout is VE_RECEIVE_PACKET_TIMEOUT.

Another type of callback is also sent for the first received packet after a dead connection to inform the user about the fact that the connection is alive again. The callback message `errCode` for this case is VE_PACKET_RECEIPT_RESTARTED.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h.

GIPSVE_SetDeadOrAliveObserver

This function installs the observer class implementation, which enables reception of periodic dead-or-alive decisions on a per-channel basis.

Syntax

```
int GIPSVE_SetDeadOrAliveObserver(GIPSVEConnectionObserver* observer);
```

Parameters

observer [in] An instance of the `GIPSVEConnectionObserver` implementation. If this pointer is set to `NULL`, the observer is removed.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h.

GIPSVE_SetPeriodicDeadOrAliveStatus

This function enables or disables the periodic dead-or-alive callback functionality for a specified channel.

Syntax

```
int GIPSVE_SetPeriodicDeadOrAliveStatus(int channel, bool enable, int
  sampleTimeSeconds = 2);
```

Parameters

channel [in] The channel ID number.

enable [in] If set to `true`, periodic dead-or-alive notification is enabled. If set to `false`, periodic dead-or-alive notification is disabled.

sampleTimeSeconds [in] Time beteen two dead-or-alive decision in seconds.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The `timeoutSeconds` value should be within the interval 1 < `sampleTimeSeconds` < 150 seconds.

To receive callback messages, the `GIPSVEConnectionObserver::OnPeriodicDeadOrAlive()` method must be implemented and the observer must be installed using `GIPSVE_SetDeadOrAliveObserver()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h.

GIPSVE_SendUDPPacket

This function handles sending a raw UDP data packet over an existing RTP or RTCP socket.

Syntax

```
int GIPSVE_SendUDPPacket(int channel, const void* data, unsigned int length, int&
  transmittedBytes, bool useRtcpSocket = false);
```

Parameters

channel [in] The channel ID number.

data [in] A pointer to an array containing the data to be sent.

length	[in] The size of the array pointed to by `data` in bytes.
transmittedBytes	[out] The number of transmitted bytes is placed in this parameter on return.
useRtcpSocket	[in] If this parameter is true the packet will be sent using the RTCP socket associated with the channel.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Remarks

No RTP or RTCP header is added to the data, only UDP/IP headers.

The UDP socket from which the channel sends RTP packets will be used unless the `useRtcpSocket` parameter is set to true.

The RTCP socket cannot be used if RTCP has been disabled for the channel.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVENetwork.h.

GIPSVERTP_RTCP

The `GIPSVERTP_RTCP` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Callbacks for RTP and RTCP events such as modified SSRC or CSRC.
- SSRC handling.
- Transmission of RTCP sender reports.
- Obtaining RTCP data from incoming RTCP sender reports.
- RTP and RTCP statistics (jitter, packet loss, RTT etc.).
- Forward Error Correction (FEC).
- RTP Keepalive for maintaining the NAT mappings associated to RTP flows.
- Writing RTP and RTCP packets to binary files for off-line analysis of the call quality.
- Inserting extra RTP packets into active audio stream.

NOTE: The `GIPSVERTP_RTCP`:: prefix is excluded for most API names throughout this chapter.

Enumerator GIPS_RTPDirections

This enumerator is used to specify what direction to record when RTP sessions are written to files. It is utilized by the `GIPSVE_StartRTPDump()`/`GIPSVE_StopRTPDump()` and `GIPSVE_RTPDumpIsActive()` APIs.

Syntax

```
enum GIPS_RTPDirections
{
    RTP_INCOMING = 0,
    RTP_OUTGOING
};
```

Enumerators

RTP_INCOMING	Incoming (received) RTP/RTCP session.
RTP_OUTGOING	Outgoing (transmitted) RTP/RTCP session.

Struct GIPS_CallStatistics

This structure is used to access statistics from RTCP reports through `GIPSVE_GetRTCPStatistics()`. The statistics are computed according to RFC 3550 under Sender and Receiver Reports. Refer to the RFC for more information.

Syntax

```
struct GIPS_CallStatistics
{
    unsigned short fractionLost;
    unsigned int cumulativeLost;
    unsigned int extendedMax;
    unsigned int jitterSamples;
    int rttMs;
    int bytesSent;
    int packetsSent;
    int bytesReceived;
    int packetsReceived;
};
```

Parameters

fractionLost	Fraction of packets lost in Q8 (a fixed-point arithmetic domain).
cumulativeLost	Total number of lost packets.

extendedMax	Extended highest sequence number received.
jitterSamples	Jitter in samples.
rttMs	Round-trip time in milliseconds.
bytesSent	Total number of bytes sent.
packetsSent	Total number of packets sent.
bytesReceived	Total number of bytes received.
packetsReceived	Total number of packets received.

Class GIPSVERTPObserver

This is a callback class for receiving messages that are related to RTP packets received by the VoiceEngine.

Syntax

```
class GIPSVERTCPObserver
{
public:
   virtual void OnIncomingCSRCChanged(const int channel, const unsigned int CSRC,
     const bool added) = 0;
   virtual void OnIncomingSSRCChanged(const int channel, const unsigned int SSRC)
     = 0;
};
```

GIPSVERTPObserver::OnincomingCSRCChanged

The VoiceEngine user should override the `GIPSVERTPObserver` method in a derived class. `OnIncomingCSRCChanged` will be called immediately after any SSRC in the incoming CSRC list is modified (added or removed). This function enables the user to keep track of contributing sources entering and leaving a conference or a PTT session.

Syntax

```
void OnIncomingCSRCChanged(const int channel, const unsigned int CSRC, const bool
   added);
```

Parameters

channel	The channel ID number.
CSRC	The received CSRC which has changed recently.
added	Set to `true` if the CSRC was added to the CSRC list and `false` if it was removed from the CSRC list.

Remarks

See RFC 3550 for details about the CSRC list.

A new callback is given for each modified CSRC for any given channel. Hence, if the CSRC list for chanel 0 contains four contributing sources, four callbacks will be generated, one for each CSRC.

The derived class is installed with the `GIPSVE_SetRTPObserver()` API.

GIPSVERTPObserver::OnincomingSSRCChanged

The VoiceEngine user should override the `GIPSVERTPObserver` method in a derived class. `OnIncomingSSRCChanged` will be called immediately after any incoming SSRC is changed.

Syntax

```
void OnIncomingSSRCChanged(const int channel, const unsigned int SSRC);
```

Parameters

channel	The channel ID number.
SSRC	The received SSRC which has changed recently.

Remarks

See RFC 3550 for details about the synchronization source, SSRC.

The derived class is installed with the `GIPSVE_SetRTPObserver()` API.

Class GIPSVERTCPObserver

This is a callback class for receiving messages that are related to RTCP packets received by the VoiceEngine. One example is RTCP APP packets which can be used in PTT scenarios.

Syntax

```
class GIPSVERTCPObserver
{
public:
   virtual void OnApplicationDataReceived(const int channel, const unsigned char
    subType, const unsigned int name, const char* data, const unsigned short
    dataLengthInBytes) = 0;
};
```

GIPSVERTCPObserver::OnApplicationDataReceived

The VoiceEngine user should override the `GIPSVERTCPObserver` method in a derived class. `OnApplicationDataReceived` will be called immediately after an application-defined RTCP packet (RTCP APP) packet arrives.

Syntax

```
void OnApplicationDataReceived(const int channel, const unsigned char subType,
  const unsigned int name, const char* data, const unsigned short
  dataLengthInBytes) = 0;
```

Parameters

channel	The channel ID number.
subtype	The subtype value (5 bits).
name	The chosen name for this set of APP packets.
data	A pointer to an array containing the application-dependent data.
dataLengthInBytes	The length of the `data` array. Is always a multiple of 32 bits.

Remarks

See RFC 3550 for details about RTCP APP: Application-defined RTCP Packet.

The derived class is installed with the `GIPSVE_SetRTCPObserver()` API.

GetInterface

Retrieves a pointer to the GIPSVERTP_RTCP sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVERTP_RTCP* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new GIPSVERTP_RTCP interface.

If the function fails, the return value is NULL.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

Release

Releases the GIPSVERTP_RTCP sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVEBase::GIPSVE_LastError().

Remarks

The number of calls to Release() should always match the number of calls to GetInterface().

When the reference count of all sub-APIs reaches zero, GIPSVoiceEngine::Delete() can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if Release() has been called too many times; however, -1 is given as return value to indicate that the number of calls to Release() does not match the number of calls to GetInterface().

Example Code

See GIPSVEBase::GIPSVE_Release().

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetSendSSRC

This function enables you to specify the RTP synchronization source identifier (SSRC) explicitly.

Syntax

```
int GIPSVE_SetSendSSRC(int channel, unsigned int ssrc);
```

Parameters

channel [in] The channel ID number.

ssrc [in] The SSRC.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVEBase::GIPSVE_LastError().

Remarks

The VoiceEngine usually generates this value. According to RFC 3550 the SSRC is generated as a random number.

This call should be performed before `GIPSVEBase::GIPSVE_StartSend()` is called.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetSendSSRC

This function extracts the RTP SSRC of a specific channel.

Syntax

```
int GIPSVE_GetSendSSRC(int channel, unsigned int& ssrc);
```

Parameters

channel [in] The channel ID number.

ssrc [out] On return, the current SSRC value.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Remarks

The SSRC corresponds either to what was explicitly set by `GIPSVE_SetSendSSRC()` or automatically generated by VoiceEngine. If VoiceEngine generated the value, it is unspecified before `GIPSVEBase::GIPSVE_StartSend()` is called.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRemoteSSRC

This function returns the SSRC of the incoming RTP packets.

Syntax

```
int GIPSVE_GetRemoteSSRC(int channel, unsigned int& ssrc);
```

Parameters

channel [in] The channel ID number.

ssrc [out] The SSRC of the incoming RTP packets will be copied to this output parameter.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRemoteCSRCs

This function returns the CSRCs of the incoming RTP packets.

Syntax

```
int GIPSVE_GetRemoteCSRCs(int channel, unsigned int arrCSRC[15]);
```

Parameters

channel [in] The channel ID number.

arrCSRC [out] The contributing sources (CSRCs) will be copied to this array.

Return Values

The number of integer values contained in the CSRC array if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

See `GIPSVEBase::GIPSVE_SetConferenceStatus()` for details on how to enable CSRC generation for audio conferences.

When conferencing is enabled, the VoiceEngine mixer can insert a list of the SSRC identifiers of the sources that contributed to the generation of particular packet into the RTP header of that packet. This list is called the CSRC list.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRemoteEnergy

This function returns the energy levels of the contributing sources for the incoming RTP packets.

Syntax

```
int GIPSVE_GetRemoteEnergy(int channel, unsigned int arrEnergy[15]);
```

Parameters

channel [in] The channel ID number.

arrEnergy [out] The energy levels (0-9) of the contributing sources will be copied to this array.

Return Values

The number of integer values contained in the energy array if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

See `GIPSVEBase::GIPSVE_SetConferenceStatus()` for details on how to enable energy levels in combination with CRSCs for audio conferences.

Each contributing source is mapped to an energy level between 0 and 9.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetRTCPStatus

This function enables or disables the transmission of RTCP reports on a specific channel.

Syntax

```
int GIPSVE_SetRTCPStatus(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] If this parameter is `true`, transmission of RTCP reports is enabled. If this parameter is `false`, transmission of RTCP reports is disabled.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The following fields are supported: Sender Reports (SR), Receiver Reports (RR), SDES:CNAME and BYE.

RTCP SR will be transmitted when sending is active.

RTCP RR will be transmitted when listening and playing are active but sending is inactive, i.e., RTP transmission is disabled. A valid destination address must also be defined for this case using the `GIPSVE_Base::GIPSVE_SetSendDestination()` API.

BYE is sent automatically when `GIPSVEBase::GIPSVE_StopSend()` is called.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRTCPStatus

This function returns the RTCP status for a specific channel.

Syntax

```
int GIPSVE_GetRTCPStatus(int channel, bool& enabled);
```

Parameters

channel [in] The channel ID number.

enabled [out] A binary reference output which is set to `true` if RTCP is enabled and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

RTCP is enabled by default for all created channels.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetRTCP_CNAME

This function sets the canonical name (CNAME) parameter for RTCP reports on a specific channel.

Syntax

```
int GIPSVE_SetRTCP_CNAME(int channel, const char* cname);
```

Parameters

channel [in] The channel ID number.

cname [in] A pointer to an array containing the CNAME as a null-terminated string.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Default name is "user1@undefined".

The `cname` string cannot be longer than 255 bytes.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRemoteRTCP_CNAME

This function retrieves the canonical name (CNAME) parameter for RTCP reports on a specific channel.

Syntax

```
GIPSVE_GetRemoteRTCP_CNAME(int channel, char* cname);
```

Parameters

channel [in] The channel ID number.

cname [out] A pointer to a character buffer to receive CNAME string.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If no RTCP report has been received, or if the report does not contain any CNAME field, this will be an empty string.

The returned string can be up to 255 bytes long.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRemoteRTCPData

This function obtains RTCP data from incoming RTCP Sender Reports.

Syntax

```
virtual int GIPSVE_GetRemoteRTCPData(int channel, unsigned int& NTPHigh, unsigned
   int& NTPLow, unsigned int& timestamp, unsigned int& playoutTimestamp, unsigned
   int* jitter = NULL, unsigned short* fractionLost = NULL);
```

Parameters

channel	[in] The channel ID number.
NTPHigh	[out] The seconds part of the 64-bit NTP timestamp will be placed here at return.
NTPLow	[out] The fractional part of the 64-bit NTP timestamp will be placed here at return.
timeStamp	[out] The RTP timestamp will be placed here at return.
playoutTimeStamp	[out] The playout timestamp at the time of the last RTCP packet arrival will be placed here at return. This is a locally obtained parameter.
jitter	[out_opt] The jitter statistics will be placed here at return, unless `jitter` is `NULL`.
fractionLost	[out_opt] The fraction of packets loss will be placed here at return, unless `fractionLost` is `NULL`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The NTP time and the corresponding RTP timestamp (`timeStamp`) can be used for video synchronization. These parameters are further explained in RFC 3550, section 6.4.1.

`playoutTimeStamp` is also provided to further facilitate video synchronization.

RTCP must be enabled for the specified channel (see `GIPSVE_SetRTCPStatus()`). If no RTCP packets have been received, all parameters will be 0. An NTP time stamp that is zero is always invalid.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRTPStatistics

This function extracts the RTP statistics for a specific channel.

Syntax

```
GIPSVE_GetRTPStatistics(int channel, unsigned int& averageJitterMs, unsigned int&
   maxJitterMs, unsigned int& discardedPackets);
```

Parameters

channel	[in] The channel ID number.
avgerageJitterMs	[out] Short-time average jitter (in milliseconds).
maxJitterMs	[out] Maximum short-time jitter (in milliseconds).
discardedPackets	[out] The number of discarded packets on a channel during the call.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The jitter parameters (`averageJitterMs` and `maxJitterMs`) are reset at each RTCP packet transmission for the given channel ID.

Packets are generally discarded due to the channel not being mixed for playout.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetRTPObserver

This function installs an instance of a `GIPSVERTPObserver` derived class.

Syntax

```
int GIPSVE_SetRTPObserver(GIPSVERTPObserver* observer);
```

Parameters

observer [in] A pointer to an instance of a `GIPSVERTPObserver` derived class. If this pointer is set to `NULL`, the observer is removed.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetRTCPObserver

This function installs an instance of a `GIPSVERTCPObserver` derived class.

Syntax

```
int GIPSVE_SetRTCPObserver(GIPSVERTCPObserver* observer);
```

Parameters

observer [in] A pointer to an instance of a `GIPSVERTCPObserver` derived class. If this pointer is set to `NULL`, the observer is removed.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SendApplicationDefinedRTCPPacket

This function sends an RTCP APP packet on a specific channel.

Syntax

```
int GIPSVE_SendApplicationDefinedRTCPPacket(int channel, const unsigned char
   subType, unsigned int name, const char* data, unsigned short
   dataLengthInBytes)
```

Parameters

channel [in] The channel ID number.

subtype [in] May be used as a subtype to allow a set of APP packets to be defined as a unique name.

name [in] A name chosen by the user defining the set of APP packets to be unique with respect to other APP packets the VoiceEngine might receive.

data [in] A pointer to an array containing the application-dependent data field of the APP packet to send.

dataLengthInBytes [in] The length of the array pointed to by `data`. Must be a multiple of 32 bits.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The input parameters are further explained in RFC 3550, section 6.7.

A valid RTCP APP packet is not transmitted directly when this function is called. Instead, the packet is scheduled for transmission and sent when the next RTCP packet is transmitted.

Sending and RTCP must be enabled before this function can be called successfully.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRTCPStatistics

These functions retrieve the RTCP statistics of a specific channel.

Syntax

```
int GIPSVE_GetRTCPStatistics(int channel, unsigned short& fractionLost, unsigned
   int& cumulativeLost, unsigned int& extendedMax, unsigned int& jitterSamples,
   int& rttMs);

int GIPSVE_GetRTCPStatistics(int channel, GIPS_CallStatistics& stats);
```

Parameters

channel [in] The channel ID number.

fractionLost [out] Fraction of packets lost in Q8 (a fixed-point arithmetic domain).

cumulativeLost [out] Total number of packets lost.

extendedMax [out] Extended maximum sequence number (as defined by RFC 3550).

jitterSamples [out] Jitter (as defined by RFC 3550), in samples.

rttMs [out] The round-trip time in milliseconds.

stats [out] A reference to a `GIPS_CallStatistics` structure to which the statistics will be copied.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The acquired statistics is based on the received/incoming RTP packet stream.

The `fractionLost`, `cumaltiveLost`, `extendedMax`, and `jitterSamples` value are all reset at each RTCP packet transmission for the given channel ID.

`rttMs` is reset when RTCP is enabled or when incoming SSRC changes for the given channel ID.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetFECStatus

This function enables or disables Forward Error Correction (FEC) on a specific channel.

NOTE: It is recommended that you use one of the GIPS robust codecs instead of FEC, since FEC adds end-to-end delay.

Syntax

```
int GIPSVE_SetFECStatus(int channel, bool enable, int redPayloadtype = -1);
```

Parameters

channel [in] The channel ID number.

enable [in] If this parameter is `true`, FEC is enabled. If this parameter is `false`, FEC is disabled.

redPayloadtype [in] The desired RED payload type. If omitted or set to -1, the default type is used.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

VoiceEngine supports FEC according to RFC 2198.

When a low bit-rate standard codec (for example, G.729 or G.723.1) is used, FEC might be beneficial. FEC uses the primary codec as a redundant payload. When enabled, RTP data is sent with the RED payload instead of the primary codec payload. For more details, and SDP information, refer to RFC 2198.

If iSAC super-wideband is used as primary codec, the current iSAC FEC only protects the wideband part of the iSAC bit-stream. In an event of packet loss, only wideband content of the frame is recovered. As PLC is not engaged for a single packetloss if FEC is activated, then, in such a case, a full recovery of the frame is not achieved. This might yield artifacts in decoded audio due to changes in audio bandwidth. In future releases this feature will be completed.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_SetRTPKeepaliveStatus

This function enables or disables an RTP keepalive mechanism which can be used to maintain an existing Network Address Translator (NAT) mapping while regular RTP is no longer transmitted.

NOTE: See Section 4.6 (RTP Packet with Unknown Payload Type) at http://www.ietf.org/internet-drafts/draft-ietf-avt-app-rtp-keepalive-04.txt for more details.

Syntax

```
int GIPSVE_SetRTPKeepaliveStatus(int channel, bool enable, int unknownPayloadType
   = 0, int deltaTransmitTimeSeconds = 15);
```

Parameters

channel	[in] The channel ID number.
enable	[in] If this parameter is `true`, RTP keepalive is enabled. If `false`, RTP keepalive is disabled.
unknownPayloadType	[in_opt] Dynamic payload type that has not been negotiated by the peers (e.g. not negotiated within the SDP offer/answer). Valid input range is [0,127].
deltaTransmitTimeSeconds	[in_opt] Specifies the time, in seconds, between two successive RTP keepalive packets. Default value is 15 seconds. Valid input range is [1,60] seconds.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

RTP keepalive packets are all of length 0 (contains no RTP payload).

RTP keepalive packets will only be transmitted when *all* of the following conditions are met:

1. RTP keepalive is enabled.
2. The VE is in a listening state.
3. A destination address is defined using the `GIPSVE_Base::GIPSVE_SetSendDestination()` API.
4. The VE is not sending or is in an on-hold state.
5. Regular RTP packets are not transmitted.

RTP keepalive packets are not transmitted in combination with enabled VAD/DTX/CNG, not even during long silence periods. Taking muted G.729AB as an example: even if the SID update rate is very low (~0.3Hz), it should be sufficient to maintain an existing NAT mapping without additional RTP keepalive packets.

GIPS VoiceEngine will silently discard incoming RTP keepalive packets.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_GetRTPKeepaliveStatus

This function returns the RTP keepalive status for a specific channel.

Syntax

```
int GIPSVE_GetRTPKeepaliveStatus(int channel, bool& enabled, int&
   unknownPayloadType, int& deltaTransmitTimeSeconds);
```

Parameters

channel	[in] The channel ID number.
enabled	[out] A binary reference output which is set to `true` if RTP keepalive is enabled (i.e., can be transmitted) and `false` otherwise.
unknownPayloadType	[out] Contains the dynamic payload type as output.
deltaTransmitTimeSeconds	[out] Contains the delta time, in seconds, between transmission of two successive RTP keepalive packets as output.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

A returned status of `true`, does not guarantee that RTP keepalive packets are actually being transmitted. All conditions given above (see `GIPSVE_SetRTPKeepaliveStatus()`) must be fulfilled before transmission starts.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_StartRTPDump

This function enables capturing of RTP packets to a binary file on a specific channel and for a given direction. The file can later be replayed using e.g. RTP Tools' `rtpplay` since the binary file format is compatible with the `rtpdump` format.

NOTE: It is recommended that you use this API for debugging purposes only since the created files can become very large.

Syntax

```
int GIPSVE_StartRTPDump(int channel, const char* fileNameUTF8, GIPS_RTPDirections
   direction = RTP_INCOMING);
```

Parameters

channel	[in] The channel ID number.
fileNameUTF8	[in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.

direction [in_opt] A `GIPS_RTPDirections` enumerator that sets the recording direction.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

It is possible to enable this functionality before any RTP media is received or transmitted. As soon as the RTP session starts, packets will be stored in the already opened file.

This API allows the user to capture RTP sessions without using an external tool like Wireshark (http://www.wireshark.org/).

Both RTP and RTCP packets are captured in both directions.

If RTP dump is activated on the incoming side, the packets are captured *after* decryption (e.g. SRTP).

If RTP dump is activated on the outgoing side, the packets are captured *before* encryption (e.g. SRTP).

See http://www.cs.columbia.edu/irt/software/rtptools/ for details on how to use the command-line tool `rtpplay`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_StopRTPDump

This function disables capturing of RTP packets to a binary file on a specific channel and for a given direction.

Syntax

```
int GIPSVE_StopRTPDump(int channel, GIPS_RTPDirections direction = RTP_INCOMING);
```

Parameters

channel [in] The channel ID number.

direction [in_opt] A `GIPS_RTPDirections` enumerator that sets the recording direction.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_RTPDumpIsActive

This function retrieves the current RTP capturing state for the specified channel and direction.

Syntax

```
int GIPSVE_RTPDumpIsActive(int channel, GIPS_RTPDirections direction =
   RTP_INCOMING);
```

Parameters

channel [in] The channel ID number.

direction [in_opt] A `GIPS_RTPDirections` enumerator that sets the recording direction.

Return Values

The return value is `0` if RTP dump is disabled and `1` if RTP dump is enabled. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android, iPhone
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVE_InsertExtraRTPPacket

This function enables sending of an extra RTP packet using an existing/active RTP session. It is possible to set the payload type, marker bit and payload of the extra RTP packet.

Syntax

```
int GIPSVE_InsertExtraRTPPacket(int channel, unsigned char payloadType, bool
   markerBit, const char* payloadData, unsigned short payloadSize);
```

Parameters

channel [in] The channel ID number.

payloadType [in] Payload type in the RTP header (7-bits).

markerBit [in] Marker bit in the RTP header (1 bit).

payloadData [in] A pointer to a data buffer containing the RTP payload.

payloadSize [in] Size (in bytes) of the payload.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The RTP sequence number is updated for each transmitted extra RTP packet.

The RTP timestamp is *not* updated for each transmitted extra RTP packet.

Inserting many consequtive extra RTP packets will have a negative impact of the audio quality.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVERTP_RTCP.h

GIPSVEVQE

The GIPSVEVQE sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Noise Suppression (NS).
- Automatic Gain Control (AGC).
- Echo Control (EC).
- Receiving side VAD.
- Measurements of instantaneous Speech, Noise and Echo levels.

NOTE: The GIPSVEVQE:: prefix is excluded for most API names throughout this chapter.

Struct GIPS_AGC_config

This structure is used to specify the AGC configuration. It is utilized by the `GIPSVE_SetAGCConfig()` and `GIPSVE_GetAGCConfig()` APIs.

Syntax

```
struct GIPS_AGC_config
{
    unsigned short targetLeveldBOv;
    unsigned short digitalCompressionGaindB;
    bool           limiterEnable;
};
```

Parameters

targetLeveldBOv

The target envelope level of the entire system, in negative decibels from overload (or digital full-range). For instance, a value of 3 corresponds to -3 dBOv, or a target envelope 3 dB below digital full-range. Valid range is [0, 31] (default is 3).

digitalCompressionGaindB	Specifies the range in gain the digital compression stage may apply, in decibels. A higher number corresponds to greater compression; a value of 0 will leave the signal uncompressed. Valid range is [0, 90] (default is 9).
limiterEnable	If enabled, the compression stage will hard limit the signal to the target level. Otherwise, the signal will be compressed but not limited above the target level. Default mode is enabled.

Enumerator GIPS_NSmodes

This enumerator is used to specify the degree of noise suppression. It is utilized by the `GIPSVE_SetNSStatus()` and `GIPSVE_GetNSStatus()` APIs.

Syntax

```
enum GIPS_NSmodes
{
    NS_UNCHANGED = 0,
    NS_DEFAULT,
    NS_CONFERENCE,
    NS_LOW_SUPPRESSION,
    NS_MODERATE_SUPPRESSION,
    NS_HIGH_SUPPRESSION,
    NS_VERY_HIGH_SUPPRESSION
};
```

Enumerators

NS_UNCHANGED	The mode set in the previous call will be used.
NS_DEFAULT	The default mode for the current platform (one of the values below).
NS_CONFERENCE	The recommended mode for a conference (NS_HIGH_SUPPRESSION).
NS_LOW_SUPPRESSION	Lowest suppression. Provides 6 dB attenuation.
NS_MODERATE_SUPPRESSION	Provides 10 dB attenuation.
NS_HIGH_SUPPRESSION	Provides 15 dB attenuation.
NS_VERY_HIGH_SUPPRESSION	Highest suppression. Provides 20 dB attenuation.

Enumerator GIPS_AGCmodes

This enumerator is used to specify the type of Automatic Gain Control (AGC). It is utilized by the `GIPSVE_SetAGCStatus()` and `GIPSVE_GetAGCStatus()` APIs.

Syntax

```
enum GIPS_AGCmodes
```

```
{
    AGC_UNCHANGED = 0,
    AGC_DEFAULT,
    AGC_ADAPTIVE_ANALOG,
    AGC_ADAPTIVE_DIGITAL,
    AGC_FIXED_DIGITAL
};
```

Enumerators

AGC_UNCHANGED	Leave the AGC mode at its current setting.
AGC_DEFAULT	The default mode for the current platform (one of the values below).
AGC_ADAPTIVE_ANALOG	Adaptive mode intended for use if an analog volume control is available on the capture device. This is the recommended mode in typical VoIP scenarios such as a PC softphone.
AGC_ ADAPTIVE_DIGITAL	Adaptive mode intended for situations in which an analog volume control is unavailable. It operates in a similar fashion to the adaptive analog mode, but with scaling applied in the digital domain. This is the recommended mode for conference servers and embedded devices (e.g. mobile and IP phones) lacking an analog volume control and where the input level is not well known.
AGC_FIXED_DIGITAL	This mode is distinguished from the adaptive modes by considering only short time-window of the input signal. It applies a fixed gain through most of the input level range, and compresses (or gradually reduces gain with increasing level) the input signal at higher levels. This mode is preferred on embedded devices where the capture signal level is predictable, so that a known gain can be applied. The adaptive modes utilize this compression stage implicitly.

Enumerator GIPS_ECmodes

This enumerator is used to specify the type of Echo Control (EC). It is utilized by the `GIPSVE_SetECStatus()` and `GIPSVE_GetECStatus()` APIs.

Syntax

```
enum GIPS_ECmodes
{
    EC_UNCHANGED = 0,
    EC_DEFAULT,
    EC_CONFERENCE,
    EC_AEC,
    EC_AES,
```

```
    EC_AECM,
    EC_NEC_IAD
};
```

Enumerators

EC_UNCHANGED	The mode set in the previous call will be used.
EC_DEFAULT	The default mode for the current platform (one of the values below).
EC_CONFERENCE	The recommended mode for a conference.
EC_AEC	Acoustic Echo Cancellation.
EC_AES	Acoustic Echo Suppression.
EC_AECM	Echo suppression for mobile devices.
EC_NEC_IAD	Network Echo Cancellation. Only supported in VoiceEngine ATA.

Remarks

In most situations AEC is recommended. It will generally provide the best performance. AES has much lower complexity and better robustness to poor device buffers. AECM is developed to have better performance on mobile devices.

Enumerator GIPS_AESmodes

This enumerator is used to specify what Acoustic Echo Suppression (AES) mode to utilize given than the EC mode is set to EC_AES. It is utilized by the `GIPSVE_SetECStatus()` and `GIPSVE_GetECStatus()` APIs.

Syntax

```
enum GIPS_AESmodes
{
    AES_DEFAULT = 0,
    AES_NORMAL,
    AES_HIGH,
    AES_ATTENUATE,
    AES_NORMAL_SOFT_TRANS,
    AES_HIGH_SOFT_TRANS,
    AES_ATTENUATE_SOFT_TRANS
};
```

Enumerators

AES_DEFAULT	The default mode for the current platform (one of the values below).
AES_NORMAL	Normal.

AES_HIGH	High echo. To be utilized when the echo is expected to be loud relative to the participant's voice.
AES_ATTENUATE	Attenuate. The AES will not fully suppress. This is most useful if the echo is expected to be quiet relative to the participant's voice.
AES_NORMAL_SOFT_TRANS	Normal with soft transition switching.
AES_HIGH_SOFT_TRANS	High with soft transition switching.
AES_ATTENUATE_SOFT_TRANS	Attenuate with soft transition switching.

Remarks

The AES normally operates as a hard switch (suppresses the signal fully or not at all). Soft transition switching provides an intermediate state for a less abrupt transition.

Class GIPSVERxVadCallback

VAD is typically used on the outgoing signal as a component of a DTX system. It may sometimes be useful to detect voice activity in the received (incoming) signal. This callback class allows for notification of received VAD status.

The VoiceEngine user should override the `OnRxVad()` method in a derived class. `OnRxVad()` will be called after any change in voice activity status on a particular channel is detected.

NOTE: Always ensure that the callback functions are kept as short as possible to ensure that additional callbacks are not delayed.

```
class GIPSVERxVadCallback
{
public:
    virtual void OnRxVad(int channel, int vadDecision) = 0;
};
```

GIPSVERxVadCallback::OnRxVad

This method will be called by the VoiceEngine after any change in voice activity status on a particular channel is detected.

Syntax

```
void OnRxVad(int channel, int vadDecision);
```

Parameters

channel	[out] The channel ID number.
vadDecision	[out] The binary output will be `0` if voice is not detected and `1` if voice is detected.

Remarks

The derived class is installed with `GIPSVE_InitRxVad()`.

The VoiceEngine generates this callback within a critical section. It is therefore recommended not to call any VoiceEngine APIs within the user-implemented callback function. Also, ensure that the callback function is kept as short as possible to prevent possible deadlocks.

GetInterface

Retrieves a pointer to the GIPSVEVQE sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEVQE* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created GIPSVoiceEngine object.

Return Values

If the function succeeds, the return value is a pointer to the new GIPSVEVQE interface.

If the function fails, the return value is NULL.

Remarks

Each call to this function increments an internal reference counter for the specified GIPSVoiceEngine object. This reference count is decreased by calling the corresponding Release() method and it must be zero when the VoiceEngine instance is deleted (see also GIPSVoiceEngine::Delete()).

Example Code

See GIPSVEBase::GIPSVE_GetInterface().

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

Release

Releases the GIPSVEVQE sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVEBase::GIPSVE_LastError().

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_SetNSStatus

This function enables or disables the Noise Suppression (NS) functionality. GIPS NS reduces noise in the microphone signal of all channels.

Syntax

```
int GIPSVE_SetNSStatus(bool enable, GIPS_NSmodes mode = NS_UNCHANGED);
```

Parameters

enable [in] If this parameter is `true`, NS is enabled. If the parameter is `false`, NS is disabled.

mode [in_opt] A `GIPS_NSmodes` enumerator that sets the degree of noise suppression.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

An NS mode, for example NS_DEFAULT, must be set the first time this function is called. If only the enable parameter is set in the following calls, the NS will be turned on / off using the previously set mode. It is also possible to save a mode without enabling NS, if the enable parameter is set to `false`.

This method affects all active channels the same way. It is not possible to apply different settings for different channels in multi-channel scenarios.

If a conference, with more than 2 participants, is set up using `GIPSVE_SetConferenceStatus()`it is recommended to use the NS_CONFERENCE mode. This mode will use a more aggressive setting for the NS.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetNSStatus

Retrieves the current Noise Suppression (NS) status and mode settings.

Syntax

```
int GIPSVE_GetNSStatus(bool& enabled, GIPS_NSmodes& mode);
```

Parameters

enabled [out] A binary reference output which is set to `true` if NS is enabled and `false` otherwise.

mode [out] A `GIPS_NSmodes` enumerator which will contain the current NS mode on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_SetAGCStatus

This function enables or disables the Automatic Gain Control (AGC) functionality on the input side for all active channels. The AGC adjusts the microphone signal to an appropriate level.

Syntax

```
int GIPSVE_SetAGCStatus(bool enable, GIPS_AGCmodes mode = AGC_UNCHANGED);
```

Parameters

enable [in] If this parameter is `true`, AGC is enabled. If the parameter is `false`, AGC is disabled.

mode [in_opt] A `GIPS_AGCmodes` enumerator that sets the type of AGC mode.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

An AGC mode, for example AGC_DEFAULT, must be set the first time this function is called. If only the enable parameter is set in the following calls, the AGC will be turned on / off using the previously set mode. It is also possible to save a mode without enabling AGC, if the enable parameter is set to `false`.

This method affects all active channels the same way. It is not possible to apply different settings for different channels in multi-channel scenarios.

In some rare cases, audio capture hardware may produce a saturated signal (i.e. so loud the audio is distorted) despite the AGC having caused the volume to be reduced as much as possible. In such an instance, VE will throw the `VE_SATURATION_WARNING` error through `CallbackOnError`. In response, if possible, the user is encouraged to disable or reduce any analog microphone boosting in the capture device's settings.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetAGCStatus

Retrieves the current Automatic Gain Control (AGC) status and mode settings.

Syntax

```
int GIPSVE_GetAGCStatus(bool& enabled, GIPS_AGCmodes& mode);
```

Parameters

enabled [out] A binary reference output which is set to `true` if AGC is enabled and `false` otherwise.

mode [out] A `GIPS_AGCmodes` enumerator which will contain the current AGC mode on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_SetAGCConfig

This function modifies the Automatic Gain Control (AGC) configuration for all active channels.

Syntax

```
int GIPSVE_SetAGCConfig(const GIPS_AGC_config config);
```

Parameters

config [in] A `GIPS_AGC_config` structure which specifies the AGC configuration.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Only use this method in situations where the working conditions are well known. This API is intended for advanced users only.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetAGCConfig

This function retrieves the Automatic Gain Control (AGC) configuration for all active channels.

Syntax

```
int GIPSVE_GetAGCConfig(GIPS_AGC_config& config);
```

Parameters

config [out] A `GIPS_AGC_config` structure which will contain the current AGC configuration on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_SetECStatus

This function enables or disables the Echo Control (EC) functionality.

Syntax

```
int GIPSVE_SetECStatus(bool enable, GIPS_ECmodes mode = EC_UNCHANGED,
   GIPS_AESmodes AESmode = AES_DEFAULT, int AESattn = 28);
```

Parameters

enable — [in] If this parameter is `true`, echo control is enabled. If the parameter is `false`, echo control is disabled.

mode — [in_opt] A `GIPS_ECmodes` enumerator that sets the type of echo control. The recommended mode is EC_DEFAULT.

AESmode — [in_opt] A `GIPS_AESmodes` enumerator that sets the type of Acoustic Echo Suppression (AES). This parameter is only utilized if mode is set to EC_AES.

AESattn — [in_opt] Sets the AES attenuation when one of the attenuate `AESmodes` is used (AES_ATTENUATE or AES_ATTENUATE_SOFT_TRANS). It can be set in the range 0 < `AESattn` < 32, where a higher number corresponds to greater attenuation. This parameter is only utilized if mode is set to EC_AES.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

An EC mode, for example EC_DEFAULT, must be set the first time this function is called. If only the enable parameter is set in the following calls, the EC will be turned on / off using the previously set mode. It is also possible to save a mode without enabling EC, if the enable parameter is set to `false`.

Acoustic echo occurs when there is an acoustic coupling between the speaker and microphone (the microphone can "hear" the speaker output). This manifests as the remote participant hearing their speech repeated back to them. The acoustic echo cancellation (AEC) and acoustic echo suppression (AES) products serve to eliminate or mitigate the echo.

Either AEC or AES will be used as selected (not both).

In most situations AEC is recommended. It will generally provide the best performance. AES has much lower complexity and better robustness to poor device buffers. AECM is developed to have better performance on mobile devices.

If a conference, with more than 2 participants, is set up using `GIPSVE_SetConferenceStatus()`it is recommended to use the EC_CONFERENCE mode. This mode will use a more aggressive setting for the AEC.

The AES normally operates as a hard switch (suppresses the signal fully or not at all). Soft transition switching provides an intermediate state for a less abrupt transition.

GIPS VE Mobile for Symbian only supports the EC_AES mode.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetECStatus

Retrieves the current Echo Control (EC) status and mode settings.

Syntax

```
int GIPSVE_GetECStatus(bool& enabled, GIPS_ECmodes& mode, GIPS_AESmodes& AESmode,
   int& AESattn);
```

Parameters

enabled [out] A binary reference output which is set to `true` if EC is enabled and `false` otherwise.

mode [out] A `GIPS_ECmodes` enumerator which will contain the current echo control mode on return.

AESmode [out] A `GIPS_AESmodes` enumerator which will contain the current AES mode on return. It will only contain a valid result if mode is set to EC_AES.

AESattn [out] The current AES attenuation is copied to this output parameter. It will only contain a valid result if mode is set to EC_AES.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_SetConfStatus

THIS FUNCTION IS DEPRECATED. Please see the EC_CONFERENCE mode for `GIPSVE_SetECStatus()` and NS_CONFERENCE mode for `GIPSVE_SetNSStatus()`.

Syntax

```
int GIPSVE_SetConfStatus(bool enable, GIPS_ConfModes mode = TWO_PARTICIPANTS);
```

Parameters

enable [in] If this parameter is `true`, the appropriate settings for the scenario indicated by mode are enabled. If this parameter is `false`, VQE settings set prior to enabling this feature are restored.

mode [in_opt] A `GIPS_ConfModes` enumerator that sets the mode to either `TWO_PARTICIPANTS` or `CONFERENCING`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This function call will override previous VQE settings made by `GIPSVE_SetECStatus()` and `GIPSVE_SetNSStatus()`.

Some of the settings provided by this function may be unavailable through any other API call.

Requirements

Supported platforms	NONE
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetConfStatus

THIS FUNCTION IS DEPRECATED. Please see the EC_CONFERENCE mode for `GIPSVE_SetECStatus()` and NS_CONFERENCE mode for `GIPSVE_SetNSStatus()`.

Syntax

```
int GIPSVE_GetConfStatus(bool& enabled, GIPS_ConfModes& mode);
```

Parameters

enabled [out] The enable mode is copied to this reference output.

mode [out] The scenario mode is copied to this reference output.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	NONE
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_InitRxVad

This function installs an instance of a `GIPSVERxVadCallback` derived class.

Syntax

```
int GIPSVE_InitRxVad(GIPSVERxVadCallback* rxVadCallback);
```

Parameters

rxVadCallback A pointer to an instance of a `GIPSVERxVadCallback` derived class. Set to NULL to disable callbacks.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux, Symbian (S60 3rd Ed.)
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_VoiceActivityIndicator

This function returns the VAD/DTX activity for the current microphone audio frame.

Syntax

```
int GIPSVE_VoiceActivityIndicator(int channel);
```

Parameters

channel	The channel ID number.

Return Values

0	Voice is not detected.
1	Voice is detected.
-1	An error occurred. A specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_SetMetricsStatus

This function enables or disables the possibility to retrieve instantaneous Speech, Noise and Echo metrics during an active call.

Syntax

```
int GIPSVE_SetMetricsStatus(bool enable);
```

Parameters

enable	[in] If this parameter is `true`, instantaneous Speech, Noise and Echo metrics are reset and enabled. If the parameter is `false`, Speech, Noise and Echo metrics are disabled.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

It is required to call this function with `true` as input argument to ensure that `GIPSVE_GetSpeechMetrics()`, `GIPSVE_GetNoiseMetrics()` and `GIPSVE_GetEchoMetrics()` gives valid output results.

The VoiceEngine must be initialized before this function is called.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetMetricsStatus

Rertieves the current Speech, Noise and Echo metric status.

Syntax

```
int GIPSVE_GetMetricsStatus(bool& enabled);
```

Parameters

enabled [out] A binary reference output which is set to `true` if instantaneous Speech, Noise and Echo metrics are enabled and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetSpeechMetrics

Retrieves the instantaneous speech level metrics for the transmitted and received signals.

Syntax

```
int GIPSVE_GetSpeechMetrics(int& levelTx, int& levelRx);
```

Parameters

levelTx [out] Contains the instantaneous speech level of the transmitted signal for all active channels.

levelRx [out] Contains the instantaneous speech level of the combined received signal for all active channels.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Metric calculation must be enabled before any valid results can be retrieved during an active call. See `GIPSVE_SetMetricsStatus()` for details.

The estimation of active speech levels follows the ITU-T P.56 specification.

The speech level metrics are reported in dBm0 and calculated as

P_S: Signal Power

P_{MAX}: Normalization power. For 16-bit digital signals, $P_{MAX} = (2^{15})^{15} = 32768^2$

Signal level = $10\log_{10}(P_S/P_{MAX}) + 6.18$ [dBm0]

The internal measurement interval is fixed and set to 1.5 seconds, i.e., the measurement frequency is ~0.67 Hz.

All measurements and calculations take place in the GIPS Voice Quality Enhancement (VQE) unit. On the sending side, it means that all signals, or actions, added before the VQE will not be included in measured transmitted speech level. Examples of actions that have no effect on the measured transmitted speech level are: muting the microphone signal, playing a file as microphone, placing a call in an on-hold state. Similarly, on the receiving side, adding a file to the output signal or scaling the output signal does not affect the measured level.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetNoiseMetrics

Retrieves the instantaneous noise level metrics for the transmitted and received signals.

Syntax

```
int GIPSVE_GetNoiseMetrics(int& levelTx, int& levelRx);
```

Parameters

levelTx [out] Contains the instantaneous noise level of the transmitted signal for all active channels.

levelRx [out] Contains the instantaneous noise level of the combined received signal for all active channels.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks
Metric calculation must be enabled before any valid results can be retrieved during an active call. See `GIPSVE_SetMetricsStatus()` for details.

The estimation of active noise levels follows the ITU-T P.56 specification.

The noise level metrics are reported in dBm0 and calculated as

P_S: Signal Power

P0: Normalization power. For 16-bit digital signals, $P0 = (2^{15})^{15} = 32768^2$

Signal level = $10\log_{10}(P_S/P0) + 6.18$ [dBm0]

The internal measurement interval is fixed and set to 1.5 seconds, i.e., the measurement frequency is ~0.67 Hz.

All measurements and calculations take place in the GIPS Voice Quality Enhancement (VQE) unit. On the sending side, it means that all signals, or actions, added before the VQE will not be included in measured transmitted speech level. Examples of actions that have no effect on the measured transmitted noise level are: muting the microphone signal, playing a file as microphone, placing a call in an on-hold state. Similarly, on the receiving side, adding a file to the output signal or scaling the output signal does not affect the measured level.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVE_GetEchoMetrics
Retrieves the instantaneous echo level metrics for the near-end and far-end signals.

Syntax
```
GIPSVE_GetEchoMetrics(int& ERL, int& ERLE, int& RERL, int& A_NLP);
```

Parameters
ERL [out] Echo Return Loss: the loss in echo power due to the acoustic and hardware system. The higher this value the lower the captured echo power will be relative to the played out power. For instance, if a headset is used, we would expect this number to be high.

ERLE	[out] Echo Return Loss Enhancement: the loss in echo power due to the echo control system. If there is echo to be removed (normally characterized by a low ERL), and the echo control is working properly, we would expect this to be high.
RERL	[out] The loss in echo power in the entire system (RERL = ERL + ERLE).
A_NLP	[out] The loss in echo power due only to the linear filtering stage of the echo canceller, and not the non-linear processing (NLP) stage.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Metric calculation must be enabled before any valid results can be retrieved during an active call. See `GIPSVE_SetMetricsStatus()` for details.

The echo control unit must be enabled before any valid results can be retrieved during an active call. See `GIPSVE_SetECStatus()` for details.

The echo level metrics are reported in dB and defined as

$ERL = 10\log_{10}(P_{far}/P_{echo})$ [dB]

$ERLE = 10\log_{10}(P_{echo}/P_{out})$ [dB]

RERL = ERL + ERLE [dB]

$A\_NLP = 10\log_{10}(P_{echo}/P_{a})$ [dB]

where

P_{far}: Far-end (played out) signal power

P_{echo}: Near-end (captured) echo signal power.

P_{out}: Signal power at the output of the echo canceller.

P_{a}: Internal signal power of the echo canceller at the point before its NLP.

The internal measurement interval is fixed and set to 1.5 seconds, i.e., the measurement frequency is ~0.67 Hz.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVQE.h

GIPSVEVolumeControl

The `GIPSVEVolumeControl` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Speaker volume controls.
- Microphone volume control.
- Non-linear speech level control.
- Mute functions.
- Additional stereo scaling methods (for Windows and Mac OS X only).

NOTE: The `GIPSVEVolumeControl`:: prefix is excluded for most API names throughout this chapter.

GetInterface

Retrieves a pointer to the `GIPSVEVolumeControl` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEVolumeControl* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEVolumeControl` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

Release

Releases the `GIPSVEVolumeControl` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values
If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks
The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code
See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetSpeakerVolume
This function sets the speaker volume level.

Syntax

```
int GIPSVE_SetSpeakerVolume(unsigned int volume);
```

Parameters
volume [in] The speaker volume level.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks
The volume level range is 0-255.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetSpeakerVolume

This function retrieves the current speaker volume.

Syntax

```
int GIPSVE_GetSpeakerVolume(unsigned int& volume);
```

Parameters

volume [out] The current speaker volume level.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The volume level range is 0-255.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetMicVolume

This function sets the microphone volume level.

Syntax

```
int GIPSVE_SetMicVolume(unsigned int volume);
```

Parameters

volume [in] The microphone volume level.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The volume level range is 0-255.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetMicVolume

This function retrieves the current microphone volume.

Syntax

```
int GIPSVE_GetMicVolume(unsigned int& volume);
```

Parameters

volume [out] The current microphone volume level.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The volume level range is 0-255.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetInputMute

This call mutes the microphone input signal completely without affecting the sound device volume.

Syntax

```
int GIPSVE_SetInputMute(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] If set to `true`, the microphone is muted. If set to `false`, the microphone is unmuted.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This function replaces the input signal with zeros. Data will still be transmitted.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard

Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetInputMute

Retrieves the current microphone input mute state.

Syntax

```
int GIPSVE_GetInputMute(int channel, bool& enabled);
```

Parameters

channel [in] The channel ID number.

enabled [out] If set to `true` at return, the microphone is muted. If set to `false`, the microphone is not muted.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetSpeechInputLevel

This function returns the microphone speech level, mapped non-linearly to the range 0 to 9. The method could be used, for instance, to display a level indicator such as this:

Syntax

```
int GIPSVE_GetSpeechInputLevel(unsigned int& level);
```

Parameters

level [out] The current microphone speech level (0 – 9).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

It is recommended to call this function every 100 ms.

Requirements

Supported platforms	Windows, MAC OS X, Linux, Symbian (S60 3rd Ed.)

VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetSpeechOutputLevel

This function returns the speaker speech level, mapped non-linearly to the range 0 to 9 for a specific channel. The method could be used, for instance, to display a level indicator such as this:

Syntax

```
int GIPSVE_GetSpeechOutputLevel(int channel, unsigned int& level);
```

Parameters

channel [in] The channel ID number.

level [out] The current speaker speech level (0 – 9).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If `channel` is set to -1, the level corresponds to all mixed channel including file playout. Otherwise it indicates the level for the specified channel only.

It is recommended to call this function every 100 ms.

Requirements

Supported platforms	Windows, MAC OS X, Linux, Symbian (S60 3rd Ed.)
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetInputLevelFullRange

This function returns the microphone speech level, mapped linearly to the range 0 to 32768.

Syntax

```
int GIPSVE_GetSpeechInputLevelFullRange(unsigned int& level);
```

Parameters

level [out] The current microphone speech level (0 – 32768).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

It is recommended to call this function every 100 ms.

The output value is always positive since the absolute max is measured.

Requirements

Supported platforms	Windows, MAC OS X, Linux, Symbian (S60 3rd Ed.)
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetOutputLevelFullRange

This function returns the speaker speech level, mapped linearly to the range 0 to 32768.

Syntax

```
int GIPSVE_GetSpeechOutputLevelFullRange(int channel, unsigned int& level);
```

Parameters

channel [in] The channel ID number.

level [out] The current speaker speech level (0 – 32768).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If `channel` is `-1`, the level corresponds to all mixed channels including file playout. Otherwise it indicates the level for the specified channel only.

It is recommended to call this function every 100 ms.

The output value is always positive since the absolute max is measured.

Requirements

Supported platforms	Windows, MAC OS X, Linux, Symbian (S60 3rd Ed.)
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetChannelOutputVolumeScaling

This function sets a volume scaling applied to the incoming signal of a specific channel.

Syntax

```
int GIPSVE_SetChannelOutputVolumeScaling(int channel, float scaling);
```

Parameters

channel [in] The channel ID number.

scaling [in] The scale factor to apply.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Remarks

Volume is usually set appropriately by the sending side. However, if there is a signal level problem it can be compensated for by this function. This scale only affects the call volume, not the volume of files played on the channel.

A scale factor from 0 to 10.0 is permitted, with a default of 1.0. Please note that a scale value greater than 1.0 makes it possible for the signal to distort, so use values greater than 1.0 with care.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetChannelOutputVolumeScaling

This function returns the current volume scale of a specific channel.

Syntax

```
int GIPSVE_GetChannelOutputVolumeScaling(int channel, float& scaling);
```

Parameters

channel [in] The channel ID number.

scaling [out] The currently applied scale factor.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), Android
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetWaveOutVolume

This function sets the Windows Wave volume.

Syntax

```
int GIPSVE_SetWaveOutVolume(unsigned int volume);
```

Parameters

volume [in] The Wave volume (0 to 255).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is supported only on Windows.

Requirements

Supported platforms	Windows (incl. CE/Mobile). NOTE - this API is not supported on Windows Vista and Windows 7 if VoE is built with support for the Core Audio API. VoE must be built for the Waveform Audio API to enable this function.
VE configuration	Standard (Waveform Audio API build)
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetWaveOutVolume

This function retrieves the current Windows Wave volume.

Syntax

```
int GIPSVE_GetWaveOutVolume(unsigned int& volume);
```

Parameters

volume [out] The current Wave volume (0 to 255).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile). NOTE - this API is not supported on Windows Vista and Windows 7 if VoE is built with support for the Core Audio API. VoE must be built for the Waveform Audio API to enable this function.
VE configuration	Standard (Waveform Audio API build)
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetOutputVolumePan

This function enables you to scale the volume of the left and right stereo channels independently.

Syntax

```
int GIPSVE_SetOutputVolumePan(float left, float right);
```

Parameters

left [in] Left channel speaker volume scale.

right [in] Right channel speaker volume scale.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The values for these parameters should be between 0 and 1.0 with 0 being totally silent.

To make this call you first need to call `GIPSVEHardware::GIPSVE_SetSoundDevices()` and set `enableStereoPlayout` to true.

This function applies the scaling to the mixed signal just before playout. This occurs after the mono signals have been panned and mixed and before they are fed into the soundcard.

This processing does not affect the volume of recorded files.

NOTE: This function is supported only on Windows, Mac OS X and LINUX ALSA.

Requirements

Supported platforms	Windows, Mac OS X, Linux ALSA
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetOutputVolumePan

This function retrieves the current left and right stereo channel scaling.

Syntax

```
int GIPSVE_GetOutputVolumePan(float& left, float& right);
```

Parameters

left [out] Current left channel speaker volume scale.

right [out] Current right channel speaker volume scale.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is supported only on Windows, Mac OS X and LINUX ALSA.

Requirements

Supported platforms	Windows, Mac OS X, LINUX ALSA
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_SetChannelOutputVolumePan

This function enables you to play out a specific channel with different volumes on the left and right stereo channels.

Syntax

```
int GIPSVE_SetChannelOutputVolumePan(int channel, float left, float right);
```

Parameters

channel [in] The channel ID number.

left [in] The left stereo channel panning factor.

right [in] The right stereo channel panning factor.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The `left` and `right` parameters should be between 0 and 1.0, specifying how much the volume should be scaled on each side, with 0 being totally silent.

To make this call you first need to call `GIPSVEHardware::GIPSVE_SetSoundDevices()` and set `enableStereoPlayout` to true.

NOTE: This function is only supported on Windows, Mac OS X and LINUX ALSA.

Requirements

Supported platforms	Windows, Mac OS X, LINUX ALSA
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

GIPSVE_GetChannelOutputVolumePan

This function gets the current panning factors for a specific channel.

Syntax

```
int GIPSVE_GetChannelOutputVolumePan(int channel, float& left, float& right);
```

Parameters

channel	[in] The channel ID number.
left	[out] The left stereo channel panning factor will be placed here on return.
right	[out] The right stereo channel panning factor will be placed here on return.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported on Windows, Mac OS X and LINUX ALSA.

Requirements

Supported platforms	Windows, Mac OS X, LINUX ALSA
VE configuration	Standard
Header	Declared in GIPSVEVolumeControl.h

Stereo Conferencing Example

On the platforms that support stereo play out the GIPSVE volume APIs can be used to place participants in a conference differently in the stereo space. This will simulate an eye-to-eye conversation with two or more other people in which the speech is heard from different directions.

The first step is to open the sound device in stereo mode. For example, if the default device is used:

```
hardware->GIPSVE_SetSoundDevices(-1, -1, false, true);
```

If the conference is to be set up between three participants, two channels need to be created for the two peers. When this is done the channel scaling and panning can be set up as follows to place the two peers in the stereo space:

```
// double gain for both peers to make sure that the
// panning doesn't make play out vol too low
volume->GIPSVE_SetChannelOutputVolumeScaling(0, 2.0);
volume->GIPSVE_SetChannelOutputVolumeScaling(1, 2.0);
// pan the channels
volume->GIPSVE_SetChannelOutputVolumePan(0, (float)0.28, (float)0.72);
volume->GIPSVE_SetChannelOutputVolumePan(1, (float)0.72, (float)0.28);
```

Then the IP addresses and ports of the peers need to be set up for the two channels respectively. When the sending and receiving has been started to the two peers they should now be panned slightly to the left and to the right. It should be noted that the two peers need to do follow the same procedure to create a 3-way conference in which all participants can hear each other and experience the stereo effect. This can be expanded to larger conferences if additional channels are used. The panning settings for 2, 3 and 4 peers are given in the following table.

	Two peers		Three peers		Four peers	
channel	left	right	left	right	left	right
0	0.28	0.72	0.20	0.80	0.17	0.83
1	0.72	0.28	0.50	0.50	0.41	0.59
2			0.80	0.20	0.59	0.41
3					0.83	0.17

The example above describes the case when all participants set up audio streams to all peers. In the case when the conference is hosted by one of the participants, see `GIPSVE_SetConferenceStatus()`, it is not possible to get the stereo effect at the peers because they receive a mixed audio stream from the host on the single channel that is set up to the host. The host can however pan all other participants.

GIPSVEHardware

The `GIPSVEHardware` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Sound device handling.
- CPU load monitoring.
- External sound card.
- Device information.

NOTE: The `GIPSVEHardware::` prefix is excluded for most API names throughout this chapter.

Class SndCardObject

This is a callback class which allows the VoiceEngine to call sound device functions handled by the user.

The VoiceEngine user should override the methods in a derived class.

Syntax

```
class SndCardObject
{
public:
   int initSpeaker();
   int initMicrophone();
   int initRecording();
   int initPlayback();
   int startRecording();
   int startPlayback();
   int stopRecording();
```

```
   int stopPlayback();
   int shutDown();
   int getMicLevel();
   int getMicLevel(int level);
   int getSpeakerLevel();
   int setSpeakerLevel(int level);
   int setDevices(unsigned int in, unsigned int out);
   ~SndCardObject() {}
protected:
   SndCardObject() {}
};
```

Remarks

Contact GIPS for more details about this class and its members.

The derived instance is installed with `GIPSVE_SetSoundCardObject()`.

GetInterface

Retrieves a pointer to the `GIPSVEHardware` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEHardware* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEHardware` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

Release

Releases the `GIPSVEHardware` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetCPULoad

This function returns the VoiceEngine's current CPU consumption in terms of the percent of total CPU availability.

Syntax

```
int GIPSVE_GetCPULoad(int& loadPercent);
```

Parameters

loadPercent [out] The VoiceEngine's CPU load.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If this value is higher than 70 % it is recommended that you do not add more channels to a conference call.

It is a known fact that he load value reported by this function is not consistent with the Windows TaskManager values. This API measures the time needed to do internal VoiceEngine processing. When this value approaches 100%, there will be problems with voice quality and delay. Note that, these problems exist even if the TaskManager reports lower CPU values. To summarize: the output result from `GIPSVE_GetCPULoad()` is better to use to determine if more conference participants can be added than the TaskManager value.

NOTE: This function is only supported on Windows.

Requirements

Supported platforms	Windows (incl. CE/Mobile),
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetSystemCPULoad

This function returns the computer's current CPU consumption in terms of the percent of total CPU availability.

Syntax

```
int GIPSVE_GetSystemCPULoad(int& loadPercent);
```

Parameters

loadPercent [out] The computer's total CPU load.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The return value for this call is the same value that can be seen in the Windows Task Manager.

NOTE: On Windows Vista, Windows 7 and on some Windows XP setups, the VoiceEngine must be run in administrator mode for this functionality to work.

Requirements

Supported platforms	Windows (incl. CE/Mobile), Symbian (S60 3rd Ed.)
VE configuration	Standard

Header	Declared in GIPSVEHardware.h

GIPSVE_GetNumOfSoundDevices

This function returns the number of devices available for playout and recording.

Syntax

```
int GIPSVE_GetNumOfSoundDevices(int& playout, int& recording);
```

Parameters

playout [out] The number of devices available for playout.

recording [out] The number of devices available for recording.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is supported only on Windows and Linux/ALSA.

Use this function to determine the maximum device index for `GIPSVE_GetRecordingDevName()`, `GIPSVE_GetPlayoutDevName()` and `GIPSVE_SetSoundDevices()`.

Linux/ALSA: Both hardware devices (i.e. card/device combinations) and virtual devices (pcms) are enumerated. Subdevices are not supported (only subdevice 0 is recognized). Some virtual devices are excluded: default (use -1 to set default device), null, and all surroundXX. Note that for a particular device to work it may have to be configured by the end user. (The end user must consult ALSA documentation.) The enumerated devices correspond to the devices retrieved with [aplay|arecord] –l and [aplay|arecord] –L.

Requirements

Supported platforms	Windows, MAC OS X and Linux/ALSA
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetRecordingDevName

This function gets the name of a specific recording device. On Windows Vista/7, it also retrieves an additional unique ID (GUID) for the recording device.

Syntax

```
int GIPSVE_GetRecordingDeviceName(int index, char* strNameUTF8, int nameLen,
  char* strGuidUTF8 = NULL, int guidLen = 0);
```

Parameters

index	[in] Device index (0, 1, 2, ..., N-1), where N is given by `GIPSVE_GetNumOfSoundDevices()`. Also -1 is a valid value and will return the name of the default recording device.
strNameUTF8	[out] A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format.
nameLen	[in] The size of the buffer pointed to by `strNameUTF8` in bytes.
strGuidUTF8	[out_opt] A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format. This parameter will only contain a unique GUID on Windows Vista or Windows 7 and when Core Audio is utilized. On other Windows versions (or when usage of Core Audio has failed), the product name in `strNameUTF8` is copied to this output parameter if it is not set to `NULL`. This parameter is ignored in Linux and OSX.
guidLen	[in_opt] The size of the buffer pointed to by `strGuidUTF8` in bytes. This parameter is only read on Windows Vista build configurations.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: Supported on Windows, Linux/ALSA and MAC OS X (not mobile platforms)

UTF-8 (8-bit UCS/Unicode Transformation Format) is a variable-length character encoding for Unicode. It is able to represent any character in the Unicode standard, yet the initial encoding of byte codes and character assignments for UTF-8 is backwards compatible with ASCII. UTF-8 encodes each character in one to four octets (8-bit bytes):

1. One byte is needed to encode the 128 US-ASCII characters.
2. Two bytes are needed for Latin letters with diacritics and for characters from Greek, Cyrillic, Armenian, Hebrew, Arabic, Syriac and Thaana alphabets.

See http://www.utf-8.com/ for more details.

Linux/ALSA: See also `GetNumOfSoundDevices()`. Hardware device names are formatted as “<Card name>, <Device name>”. Virtual device names are formatted as “<Name>” or as “<Name> (<Card name>[, <Device name>])” for virtual devices associated with a card (and device). Card and device names are replaced by their numbers if not possible to get.

Example Code

```
int idx, nRec = 0, nPlay = 0;
char devName[64] = {0};
char guidName[100] = {0};

// acquire sub-API (assuming GIPSVoiceEngine object exists)
GIPSVEHardware* hardware = GIPSVEHardware::GetInterface(ve);

// read number of available playout and recording devices
```

```
hardware->GIPSVE_GetNumOfSoundDevices(nPlay, nRec);

// display names (and GUID on Vista) for all recording devices
for (idx = 0; idx < nRec; idx++)
{
    // non Vista/7: devName is copied to guidName
    // Vista/7    : devName is the friendly name and GUID is a unique identifier
    hardware->GIPSVE_GetRecordingDeviceName(idx, devName, 64, guidName, 100));
    printf("GetRecordingDeviceName(%d) => name=%s, guid=%s\n", idx, devName, guidName);
}

// release sub-API
hardware->Release();
```

Example Output

The example above can generate the following example output on a Windows Vista/7 machine with two different recording devices:

```
GetRecordingDevName(0) => name=Microphone (SoundMAX Integrated Digital Audio),
   guid={0.0.1.00000000}.{841fccdc-7265-46e1-9c23-8fc2789be207}

GetRecordingDevName(1) => name=Line In (SoundMAX Integrated Digital Audio),
   guid={0.0.1.00000000}.{ed427c5f-0ad9-4e5f-940b-7d89ddcfceab}
```

On other Windows versions, the `guid` output would be identical to the `name` output; hence a unique product enumeration would not be possible.

Example output on Linux/ALSA with one hardware device and two virtual devices:

```
GetRecordingDevName(0) => name=Intel ICH6, Intel ICH6, guid=

GetRecordingDevName(1) => name=front (Intel ICH6, Intel ICH6), guid=

GetRecordingDevName(2) => name=pulse, guid=
```

Requirements

Supported platforms	Windows, Linux/ALSA, MAC OS X
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetPlayoutDevName

This function gets the name of a specific playout device. On Windows Vista/7, it also retrieves an additional unique ID (GUID) for the playout device.

Syntax

```
int GIPSVE_GetPlayoutDeviceName(int index, char* strNameUTF8, int nameLen, char*
   strGuidUTF8 = NULL, int guidLen = 0);
```

Parameters

index	[in] Device index (0, 1, 2, ..., N-1), where N is given by `GIPSVE_GetNumOfSoundDevices()`. Also -1 is a valid value and will return the name of the default playout device.
strNameUTF8	[out] A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format.
nameLen	[in] The size of the buffer pointed to by `strNameUTF8` in bytes.
strGuidUTF8	[out_opt] A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format. This parameter will only contain a unique GUID on Windows Vista or Windows 7 and when Core Audio is utilized. On other Windows versions (or when usage of Core Audio has failed), the product name in `strNameUTF8` is copied to this output parameter if it is not set to `NULL`. This parameter is ignored in Linux and OSX.
guidLen	[in_opt] The size of the buffer pointed to by `strGuidUTF8` in bytes. This parameter is only read on Windows Vista build configurations.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: Supported on Windows, Linux/ALSA and MAC OS X (not mobile platforms)

See `GIPSVE_GetRecordingDevName()` for more information on the UTF-8 format and Linux/ALSA notes.

Example Code

See `GIPSVE_GetRecordingDevName()`.

Requirements

Supported platforms	Windows, Linux/ALSA and MAC OS X
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_SetSoundDevices

This function sets the sound devices to be used during the call.

Syntax

```
int GIPSVE_SetSoundDevices(int recordingIndex, int playoutIndex, bool
   disableMicBoost = false, GIPS_StereoChannel recordingChannel =
   GIPS_StereoBoth)= 0;
```

Parameters

recordingIndex	[in] The sound input device identifier.

playoutIndex	[in] The sound output device identifier.
disableMicBoost	[in_opt]: Enables microphone boost if set to `false`. Disables microphone boost if set to `true`.
recordingChannel	[in_opt] A `GIPS_StereoChannel` enumerator that sets what channel to record from.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If the device index is set to `–1` the default device is used.

Windows: The device identifiers proceed incrementally from 0 (0, 1, 2, ...). The maximum index can be determined through `GIPSVE_GetNumOfSoundDevices()`. The default sound device is the Windows preferred device (WAVE_MAPPER). On Windows version that supports "Default Communications Device" that is the default device. If you still want to use the regular default device, use -2 as device index.

`disableMicBoost` works on most, but not all, sound cards. Microphone boost is presented differently on different sound cards.

The parameter `recordingChannel` enables you to record only from the left or the right recording channel when your sound card supports stereo recording.

Linux/ALSA: See also `GIPSVE_GetNumOfSoundDevices()` and `GIPSVE_GetRecordingDevName()`. For hardware devices, plughw will be used for opening the pcm (plughw:<card>,<device>) and hw for opening the mixer (hw:<card>). For virtual devices associated with a card, the virtual device name will be used for opening the pcm and hw will be used for opening the mixer (hw:<card>). For virtual devices not associated with a card, the name will be used for opening the pcm and mixer.

The default device is "default".

The following parameters are ignored on this platform:

- `disableMicBoost`
- `recordingChannel`

Linux/OSS: The parameters **recordingIndex** and **playoutIndex** must be the same on this platform. The default sound device is /dev/dsp. Setting the sound device to value X will select /dev/dspX.

The following parameters are ignored on this platform:

- `disableMicBoost`
- `recordingChannel`

Mac OSX: This call accepts both the device CoreAudio Device ID and the device index (0, 1, 2, ..., N-1), where N is given by `GIPSVE_GetNumOfSoundDevices()`. The following parameters are ignored on this platform:

- `disableMicBoost`
- `recordingChannel`

Requirements

Supported platforms	Windows, MAC OS X, Linux (see Remarks section above for limitations)
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_SetSoundDevices (Linux/ALSA)

This function sets the sound device to be used during the call for Linux/ALSA platforms.

Syntax

```
int GIPSVE_SetSoundDevices(const char* recordingDevice, const char*
   playoutDevice);
```

Parameters

recordingDevice [in] A pointer to an array containing the sound input device as a null-terminated string.

playoutDevice [in] A pointer to an array containing the sound output device as a null-terminated string.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function works only for Linux/ALSA.

The default device is "default".

This function does not check validness of the device names. If setting an invalid device name, `GIPSVE_StartPlayout`() or `GIPSVE_StartSend`() will return error when trying to open the device.

Requirements

Supported platforms	Linux/ALSA
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetPlayoutDeviceStatus

This function checks if the sound card is available to be opened for playout.

```
int GIPSVE_GetPlayoutDeviceStatus(bool& isAvailable);
```

Parameters

isAvailable [out] A binary reference output which is set to `true` if the sound card is available to be opened for playout and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetRecordingDeviceStatus

This function checks if the sound card is available to be opened for recording.

```
int GIPSVE_GetRecordingDeviceStatus(bool& isAvailable);
```

Parameters

isAvailable [out] A binary reference output which is set to `true` if the sound card is available to be opened for recording and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_ResetSoundDevice

See the integration notes for the respective platform for a description of this function.

```
int GIPSVE_ResetSoundDevice();
```

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows CE/Mobile, iPhone
VE configuration	Standard

Header	Declared in GIPSVEHardware.h

GIPSVE_SetSoundCardObject

This function installs a `SndCardObject` derived instance.

NOTE: This function requires a special VoiceEngine build configuration where the user implements the soundcard interface. Contact GIPS for more information about external soundcard configuration.

Syntax

```
int GIPSVE_SetSoundCardObject(SndCardObject& object);
```

Parameters

object [in] The derived instance.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The VoiceEngine must be initialized when calling this API, i.e., a successful call to `GIPSVEBase::GIPSVE_Init()` must have been performed.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration.
Header	Declared in GIPSVEHardware.h. The SndCardObject class is declared in a separate header file.

GIPSVE_NeedMorePlayData

This function drives the VoiceEngine and is called to obtain playout data.

NOTE: This function requires a special VoiceEngine build configuration where the user implements the soundcard interface. Contact GIPS for more information about external soundcard configuration.

Syntax

```
int GIPSVE_NeedMorePlayData(short* speechData10ms, int samplingFreqHz, int
  currentDelayMs, int encoding, int& samplesOut);
```

Parameters

speechData10ms [out] A pointer to an array to which a 10 ms frame of audio will be copied.

samplingFreqHz [in] The sampling frequency of the audio, in Hz (8000, 16000 or 48000).

currentDelayMs [in] An estimate of the delay (in milliseconds) from the time that the audio is recorded until it is handed to the VoiceEngine. This estimate is important to the echo canceller.

encoding [in] The desired encoding:
0: linear (no encoding)
1: G711 µ-law (valid only for `samplingFreqHz` = 8000)
2: G711 A-law (valid only for `samplingFreqHz` = 8000)

samplesOut [out] The number of samples in the array pointed to by **speechData10ms**.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError().`

Remarks

This function must be called every 10 milliseconds.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration.
Header	Declared in GIPSVEHardware.h. The SndCardObject class is declared in a separate header file.

GIPSVE_RecordedDataIsAvailable

This function drives the VoiceEngine, and is called to deliver recorded data to the VoiceEngine for transmission.

NOTE: This function requires a special VoiceEngine build configuration where the user implements the soundcard interface. Contact GIPS for more information about external soundcard configuration.

Syntax

```
int GIPSVE_RecordedDataIsAvailable(short* speechData10ms, int samplingFreqHz, int
   currentDelayMs, int decoding, int& encodingDone);
```

Parameters

speechData10ms [out] A pointer to an array containing a 10 ms frame of audio.

samplingFreqHz [in] The sampling frequency of the audio, in Hz (8000, 16000 or 48000).

currentDelayMs [in] An estimate of the delay (in milliseconds) from the time that the audio is recorded until it is handed to the VoiceEngine. This estimate is important to the echo canceller.

decoding [in] The desired encoding:
0: linear (no encoding)

	1: G711 µ-law (valid only for `samplingFreqHz` = 8000) 2: G711 A-law (valid only for `samplingFreqHz` = 8000)
encodingDone	[out] Information about the encoding of the current audio frame is copied to the pointee: `0`: No encoding was done. `1`: Encoding was done. `-1`: No information available.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This function must be called every 10 milliseconds.

The `encodingDone` parameter is useful when using two or more VoiceEngine instances and the codec operates on 20 ms or higher blocks. The information can be used to avoid the complexity peaks due to simultaneous encoding. This information is currently only available when using iLBC codec.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration.
Header	Declared in GIPSVEHardware.h. The SndCardObject class is declared in a separate header file.

GIPSVE_GetBuild

This function copies a date, time and mode information string for the VoiceEngine build to the provided buffer.

```
int GIPSVE_GetBuild(char* build, unsigned int length);
```

Parameters

build	[out] A pointer to an array to which the build information will be copied as a null-terminated string.
length	[out] The size of the array pointed to by `build` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is DEPRECATED.

A buffer size of 128 characters is sufficient for this call.

Requirements

Supported platforms	None
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetDevice

This function copies a device information string to the provided buffer.

```
int GIPSVE_GetDevice(char* device, unsigned int length);
```

Parameters

device [out] A pointer to an array to which the device information will be copied as a null-terminated string.

length [out] The size of the array pointed to by `device` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported on Windows CE/Mobile and Symbian.

A buffer size of 128 characters is sufficient for this call.

Requirements

Supported platforms	Windows CE/Mobile, Symbian (S60 3rd Ed.)
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetPlatform

This function copies a platform information string to the provided buffer.

```
int GIPSVE_GetPlatform(char* platform, unsigned int length);
```

Parameters

platform [out] A pointer to an array to which the build information will be copied as a null-terminated string.

length [out] The size of the array pointed to by `platform` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported on Windows CE/Mobile.

A buffer size of 128 characters is sufficient for this call.

Requirements

Supported platforms	Windows CE/Mobile
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_GetOS

This function copies an Operating System (OS) information string to the provided buffer.

```
int GIPSVE_GetOS(char* os, unsigned int length);
```

Parameters

os [out] A pointer to an array to which the OS information will be copied as a null-terminated string.

length [out] The size of the array pointed to by `os` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported on Windows CE/Mobile.

A buffer size of 128 characters is sufficient for this call.

Requirements

Supported platforms	Windows CE/Mobile
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_SetLoudspeakerStatus

This function enables/disables the loudspeaker used for music play out and speaker phone calls on mobile devices.

```
int GIPSVE_SetLoudspeakerStatus(bool enable);
```

Parameters

enable [in] If this parameter is `true`, the loudspeaker is enabled. If the parameter is `false`, the loudspeaker is disabled.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported on Windows CE/Mobile.

Requirements

Supported platforms	Windows CE/Mobile, Android
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_SetGrabPlayout

This function grabs the sound device playout stream. This can be useful on Win98 systems where you want to prevent other applications from grabbing the stream while the VoiceEngine is running.

Syntax

```
int GIPSVE_SetGrabPlayout(bool enable);
```

Parameters

enable [in] Releases the playout stream if set to `false`. Grabs the playout stream if set to `true`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_SetGrabRecording

This function grabs the sound device recording stream. This can be useful on Win98 systems where you want to prevent other applications from grabbing the stream while the VoiceEngine is running.

Syntax

```
int GIPSVE_SetGrabRecording(bool enable);
```

Parameters

enable	[in] Releases the recording stream if set to `false`. Grabs the recording stream if set to `true`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEHardware.h

GIPSVE_SetSamplingRate

This function sets the soundcard sampling rate.

Syntax

```
int GIPSVE_SetSamplingRate(int freqkHz);
```

Parameters

freqkHz	[in] The input/output sampling rate in kiloherz.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported for special Embedded Linux builds.

Supported rates are 8, 16 and 48 (default).

Requirements

Supported platforms	Embedded Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEHardware.h

GIPSVE_GetSamplingRate

This function retrieves the current soundcard sampling rate.

Syntax

```
int GIPSVE_GetSamplingRate(int& freqkHz);
```

Parameters

freqkHz [out] The current input/output sampling rate in kiloherz.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE: This function is only supported for special Embedded Linux builds.

Default sampling rate is 48 kHz.

Requirements

Supported platforms	Embedded Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEHardware.h

GIPSVEDTMF

The `GIPSVEDTMF` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Telephone event transmission.
- DTMF tone generation.
- Telephone event detection (optional).

NOTE: The `GIPSVEDTMF`:: prefix is excluded for most API names throughout this chapter. Telephone events include, but are not limited to, DTMF. See RFC 2833 for more information. Older VoiceEngine versions only supported DTMF, and for API compatibility reasons the name DTMF is kept unchanged in the sub-API and function calls. However, the functionality in this sub-API covers all telephone events where applicable. (See the description of each function call.)

Class GIPSVEDTMFCallback

This is a callback class for notification of telephone events.

The VoiceEngine user should override the `OnDTMFEvent()` method in a derived class. `OnDTMFEvent()` will be called after the detection of a telephone event.

NOTE: This is an optional feature. It relies on a GIPS product that may not be included in your VoiceEngine configuration.

Syntax

```
class GIPSVEDTMFCallback
```

```
{
public:
    virtual OnDTMFEvent(short channel, short event, bool endOfEvent) = 0;
};
```

Parameters

channel	[out] The channel that the incoming telephone event occurred on.
event	[out] Specifies the telephone event (according to RFC 2833).
endOfEvent	[out] Is set to `false` if the callback is given for the first packet in an out-of-band event and set to `true` if the callback is given for the last packet in an out-of-band event.

Remarks

The derived class is installed with `GIPSVE_SetDTMFDetection()`.

DTMF detection might not work properly if the tones are sent in-band using a low bit rate codec.

Each event will result in two callbacks (start and stop) when out-of-band detection is activated.

The `endOfEvent` flag can only be set to `true` for out-of-band DTMF detection. For all other detection modes, `endOfEvent` will always be set to `false`.

Enumerator GIPS_TelephoneEventDetectionMethods

This enumerator is used to specify the telephone event detection method. It is utilized by the `GIPSVE_SetDTMFDetection()` API.

Syntax

```
enum GIPS_TelephoneEventDetectionMethods
{
    IN_BAND = 0,
    OUT_OF_BAND,
    IN_AND_OUT_OF_BAND
};
```

Enumerators

IN_BAND	Events (tones) are detected in-band, in the audio signal itself. Only DTMF tones are detected, no other telephone events. The detection is done after out-of-band DTMF reconstruction, which means that DTMF events sent out-of-band will be detected also using this method.
OUT_OF_BAND	Out-of-band events are detected. All telephone events are detected. Note that, two callbacks will be created for each event: the first one when the event starts and the second one when the event ends.

IN_AND_OUT_OF_BAND	Both in-band and out-of-band events are detected. Note that DTMF events received out-of-band will be detected twice using this method, and two callbacks will be given (start and stop) for each detected event.

GetInterface

Retrieves a pointer to the GIPSVEDTMF sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEDTMF* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created GIPSVoiceEngine object.

Return Values

If the function succeeds, the return value is a pointer to the new GIPSVEDTMF interface.

If the function fails, the return value is NULL.

Remarks

Each call to this function increments an internal reference counter for the specified GIPSVoiceEngine object. This reference count is decreased by calling the corresponding Release() method and it must be zero when the VoiceEngine instance is deleted (see also GIPSVoiceEngine::Delete()).

Example Code

See GIPSVEBase::GIPSVE_GetInterface().

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEDTMF.h

Release

Releases the GIPSVEDTMF sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVEBase::GIPSVE_LastError().

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEDTMF.h

GIPSVE_SendDTMF

This function sends telephone events either in-band or out-of-band.

Syntax

```
int GIPSVE_SendDTMF(int channel, int eventNumber, bool outBand = true, int
   lengthMs = 160, int attenuationDb = 10);
```

Parameters

channel	[in] The channel ID number.
eventNumber	[in] Specifies the telephone event to be sent (according to RFC 2833).
outBand	[in_opt] Determines how the tone will be sent: `outBand` = `true`: Out-band tone (contained in the RTP packet) `outBand` = `false`: In-band tone (embedded into the signal payload)
lengthMs	[in_opt] The tone length in milliseconds (100 – 400 ms for DTMF, no limit for non-DTMF events).
attenuationDb	[in_opt] The tone level (volume) in negative dBm0. Allowed values are between 0 and 36: `attenuationDb` = 0 gives maximum volume (0 dBm0) `attenuationDb` = 36 gives minimum volume (-36 dBm0)

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Events are queued on the sending side to ensure that the length and spacing is correct. This means that this call can be made several times in a for-loop.

During transmission of out-of-band telephone events, no voice data is transmitted.

For in-band transmission, only DTMF events are supported.

For non-DTMF events, the length can be any value.

See RFC 2833 for more details.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEDTMF.h

GIPSVE_SetDTMFPayload

This function sets the dynamic payload number that should be used for telephone events.

Syntax

```
int GIPSVE_SetSendDTMFPayloadType(int channel, int type);
```

Parameters

channel [in] The channel ID number.

type [in] The telephone event dynamic payload type.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This call is only relevant when sending telephone events out-of-band. It is only used for setting the payload type of telephone events for sending.

To set the payload type for receiving, use `GIPSVECodec::GIPSVE_SetRecPayloadType()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEDTMF.h

GIPSVE_SetDTMFFeedbackStatus

This function enables DTMF feedback: when a DTMF tone is sent, the same tone is played out on the speaker.

Syntax

```
int GIPSVE_SetDTMFFeedbackStatus(bool enable, bool directFeedback = false);
```

Parameters

enable — [in] If this parameter is `true`, DTMF feedback is enabled. If the parameter is `false`, DTMF feedback is disabled.

directFeedback — [in_opt] Determines the feedback type:
`false`: The local feedback tone is played when the tone is sent.
`true`: The local feedback tone is played immediately, irrespective of when it is sent. The microphone will be muted during transmission to ensure that any feedback echo will not distort any sending tones.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEDTMF.h

GIPSVE_GetDTMFFeedbackStatus

This function returns the DTMF feedback status.

Syntax

```
int GIPSVE_GetDTMFFeedbackStatus(bool& enabled, bool& directFeedback);
```

Parameters

enabled — [out] A binary reference output which is set to `true` if DTMF feedback is enabled and `false` otherwise.

directFeedback — [out] A binary reference output. See `GIPSVE_SetDTMFFeedbackStatus()` for details.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVDTMF.h

GIPSVE_PlayDTMFTone

This function plays a DTMF feedback tone (only locally).

Syntax

```
int GIPSVE_PlayDTMFTone(int eventNumber, int lengthMs = 200, int attenuationDb =
   10);
```

Parameters

eventNumber [in] Specifies the DTMF tone to be sent (according to RFC 2833).

lengthMs [in] The tone length in milliseconds. Allowed values are 10 ms and higher.

attenuationDb [in_opt] The tone level (volume) in negative dBm0. Allowed values are between 0 and 36:
`attenuationDb` = 0 gives maximum volume (0 dBm0)
`attenuationDb` = 36 gives minimum volume (-36 dBm0)

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This call is normally used if DTMF tones are transmitted using the SIP/NOTIFY method. To generate other kinds of tones, or sounds in general, use `GIPSVEFile::GIPSVE_StartPlayingFileLocally()` or `GIPSVEFile::GIPSVE_StartPlayingFileAsMicrophone()`.

At least one channel must have started playout for this function to work.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVDTMF.h

GIPSVE_SetDTMFDetection

This function installs an instance of a `GIPSVE_DTMFCallback` derived class.

NOTE: This function requires a special VoiceEngine build configuration where telephone event detection is supported. Contact GIPS for more information about telephone event detection.

Syntax

```
int GIPSVE_SetDTMFDetection(bool enable, GIPSVEDTMFCallback* dtmfCallback,
   GIPS_TelephoneEventDetectionMethods detectionMethod = IN_BAND);
```

Parameters

enable [in] Enables DTMF callbacks if set to `true`. Disables DTMF callbacks if set to `false`.

dtmfCallback [in] Pointer to an instance of a `GIPSVEDTMFCallback` derived class.

detectionMethod [in] Selects the detection method, see remarks below.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Either in-band, out-of-band or both methods can be used for detection. The in-band detection is done after the reconstruction of out-of-band DTMF events, which means that they will be detected using any of the methods. If both methods are enabled, they will be detected twice. Non-DTMF events are not played out. See also `GIPS_TelephoneEventDetectionMethods`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone
VE configuration	Requires a special VoiceEngine build configuration.
Header	Declared in GIPSVDTMF.h

GIPSVE_SetDTMFPlayoutStatus

This function enables or disables DTMF tone playout for received DTMF telephone events out-of-band.

Syntax

```
int GIPSVE_SetDTMFPlayoutStatus(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] Enables playout of DTMF tones for received DTMF telephone events out-of-band if set to `true`. Disables if set to `false`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Non-DTMF telephone events are never played out.

In-band DTMF tones are not affected by this function.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android
VE configuration	Requires a special VoiceEngine build configuration.
Header	Declared in GIPSVDTMF.h

GetDTMFPlayoutStatus

This function returns the DTMF playout status.

Syntax

```
int GIPSVE_GetDTMFPlayoutStatus(int channel, bool& enabled);
```

Parameters

channel [in] The channel ID number.

enabled [out] Set to `true` if enabled, or `false` if disabled.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Android
VE configuration	Requires a special VoiceEngine build configuration.
Header	Declared in GIPSVDTMF.h

GIPSVEFile

The `GIPSVEFile` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- File playback.
- File recording.
- File conversion.
- Non-realtime RTP-to-file conversion.

NOTE: The `GIPSVEFile`:: prefix is excluded for most API names throughout this chapter.

Enumerator GIPS_FileFormats

This enumerator is used to specify the type of file format.

Syntax

```
enum GIPS_FileFormats
{
    FILE_PCM_16KHZ = 0,
    FILE_WAV = 1,
```

```
    FILE_COMPRESSED = 2,
    FILE_PCM_8KHZ = 3
};
```

Enumerators

FILE_PCM_16KHZ	PCM 16 bits/sample, mono, 16 kHz sampling rate.
FILE_WAV	PCM or u/A-law, 8/16 bits/sample, mono/stereo, 8, 11, 16, 22, 32, 44, 48 kHz sampling rates.
FILE_COMPRESSED	compressed with either iLBC or AMR.
FILE_PCM_8KHZ	PCM 16 bits/sample, mono, 8 kHz.

Remarks

The specification of the compressed file formats can be found in RFC 3267 for AMR and in RFC 3952 for iLBC.

The FILE_WAV enumerator must be used for wave files to indicate that the file contains wave headers.

NOTE: The codec must be enabled in VoiceEngine to be used for file compression.

Recording and Conversion Formats

Different compression types can be applied to the file recording and conversion APIs in this section (see e.g. `GIPSVE_StartRecordingPlayout()`and `GIPSVE_ConvertPCMToCompressed()`). Compression type is determined by the `compression` parameter as described in the following table (refer to `GIPS_CodecInst` for more details). Members of `compression` that are not specified here do not affect the recording or conversion.

Compression Type	Parameter Settings
No compression (16 kHz PCM samples)	• Set `compression` as **NULL** or omit the parameter.
ILBC compression (as specified in RFC 3952)	• Set `plname` to **"ilbc"**. • Set `plfreq` to **8000**. • Set `pacsize` to either **160** or **240** samples.
AMR compression (as specified in RFC 3267)	• Set `plname` to **"amr"**. • Set `rate` to the desired value. • Set `plfreq` to **8000**. • Set `pacsize` to **160**. NOTE: AMR must be included in VoiceEngine for this mode to be supported.
WAV files (uncompressed)	• Set `plname` to **"L16"**.

	• Set `plfreq` to either 8000 or 16000.
WAV files (G.711 u-law)	• Set `plname` to **"pcmu"**. • Set `plfreq` to 8000.
WAV files (G.711 A-law)	• Set `plname` to **"pcma"**. • Set `plfreq` to 8000.

GetInterface

Retrieves a pointer to the `GIPSVEFile` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEFile* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEFile` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

Release

Releases the `GIPSVEFile` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StartPlayingFileLocally

This function plays and mixes files with the local speaker signal for playout. Use one of the following two interfaces to play a file or stream respectively.

Syntax

```
int GIPSVE_StartPlayingFileLocally(int channel, const char* fileNameUTF8, bool
   loop = false, GIPS_FileFormats format = FILE_PCM_16KHZ, float volumeScaling =
   1.0,int startPointMs = 0, int stopPointMs = 0);

GIPSVE_StartPlayingFileLocally(int channel, InStream* stream, GIPS_FileFormats
   format = FILE_PCM_16KHZ, float volumeScaling = 1.0, int startPointMs = 0, int
   stopPointMs = 0);
```

Parameters

channel	[in] The channel ID number.
stream	[in] A pointer to an `InStream` derived instance.
fileNameUTF8	[in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.
loop	[in] `false`: The file is played through once. `true`: Plays the file repeatedly until a stop call is made.
format	[in] A `GIPS_FileFormats` enumerator that specifies the file type.

GLOBAL IP SOLUTIONS

volumeScaling	[in] Down-scales the amplitude of the signal to decrease the volume of a played file. This parameter must be between 0 and 1.0, where 1.0 is no down-scaling.
startPointMs	[in] Specifies the start of playout (in milliseconds). The interval must be at least 10 ms long and not longer than length of the file specified.
stopPointMs	[in] Specifies the stop of playout (in milliseconds). The interval must be at least 10 ms long, and not longer than the length of the file specified.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If no `startPointMs` and `stopPointMs` is specified, then by default the file is played from start to end, or until `GIPSVE_StopPlayingFileLocally()` or `GIPSVE_StopPlayout()` are called.

One channel can be used both for a call and file playback at the same time. The channel must be playing out when making this function call.

A channel can only handle one file playback at a time.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StopPlayingFileLocally

This function stops the playback of a file on a specific channel.

Syntax

```
int GIPSVE_StopPlayingFileLocally(int channel);
```

Parameters

channel	[in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_IsPlayingFileLocally

This function returns whether a channel is currently playing a file.

Syntax

```
int GIPSVE_IsPlayingFileLocally(int channel) = 0;
```

Parameters

channel [in] The channel ID number.

Return Values

0 Channel is not currently playing a file.

1 Channel is currently playing a file.

–1 An error occurred.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ScaleLocalFilePlayout

This function changes the volume scaling for a speaker file that is already playing.

Syntax

```
int GIPSVE_ScaleLocalFilePlayout(int channel, float scale);
```

Parameters

channel [in] The channel ID number.

scale [in] Down-scales the amplitude of the signal to decrease the volume of the file. This parameter must be between 0 and 1.0, where 1.0 is no down-scaling.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StartPlayingFileAsMicrophone

This function is used to read data from a file (or other location) and transmit the data either mixed with or instead of the microphone signal. Use one of the following two interfaces to play a file or stream respectively.

Syntax

```
int GIPSVE_StartPlayingFileAsMicrophone(int channel, const char* fileNameUTF8,
   bool loop = false , bool mixWithMicrophone = false, GIPS_FileFormats format =
   FILE_PCM_16KHZ, float volumeScaling = 1.0) = 0;

int GIPSVE_StartPlayingFileAsMicrophone(int channel, InStream* stream, bool
   mixWithMicrophone = false, GIPS_FileFormats format = FILE_PCM_16KHZ, float
   volumeScaling = 1.0);
```

Parameters

channel	[in] The channel ID number.
stream	[in] A pointer to an `InStream` derived instance.
fileNameUTF8	[in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.
loop	[in] `false`: The file is played through one time only. `true`: Plays the file repeatedly until a stop call is made.
mixWithMicrophone	[in] `false`: Replace the microphone signal with the file. `true`: Mix the microphone signal with the file.
format	[in] A `GIPS_FileFormats` enumerator that specifies the file type.
volumeScaling	[in] Down-scales the amplitude of the signal to decrease the volume of the file. This parameter must be between 0 and 1.0, where 1.0 is no down-scaling.

Remarks

The file will be played until the end unless `GIPSVE_StopPlayingFileAsMicrophone()` or `GIPSVE_StopPlayout()` are called

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StopPlayingFileAsMicrophone

This function stops the playback of a file as microphone signal for a specific channel.

Syntax

```
int GIPSVE_StopPlayingFileAsMicrophone(int channel);
```

Parameters

channel [in] The channel ID number.

Remarks

If the channel is set to `-1`, this function stops playing microphone files on all channels.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_IsPlayingFileAsMicrophone

This function returns whether the channel is currently playing a file as microphone signal.

Syntax

```
int GIPSVE_IsPlayingFileAsMicrophone(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

0 The channel is not currently playing a microphone file.

1 The channel is currently playing a microphone file.

–1 An error occurred.

Remarks

If `channel` is set to `-1`, this function return `1` if a file is currently playing to all channels and `0` otherwise.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ScaleFileAsMicrophonePlayout

This function changes the volume scaling for a microphone file that is already playing.

Syntax

```
int GIPSVE_ScaleFileAsMicrophonePlayout(int channel, float scale);
```

Parameters

channel [in] The channel ID number.

scale [in] Down-scales the amplitude of the signal to decrease the volume of the file. This parameter must be between 0 and 1.0, where 1.0 is no down-scaling.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StartRecordingPlayout

This function starts the recording of playout. Use one of the following two interfaces to record to a file or stream respectively.

Syntax

```
int GIPSVE_StartRecordingPlayout(int channel, const char* fileNameUTF8,
   GIPS_CodecInst* compression = NULL, int maxSizeBytes = -1);

int GIPSVE_StartRecordingPlayout(int channel, OutStream* stream, GIPS_CodecInst*
   compression = NULL);
```

Parameters

channel [in] The channel ID number.

fileNameUTF8 [in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.

stream [in] A pointer to an `OutStream` derived instance.

compression [in_opt] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for recording.

maxSizeBytes [in_opt] The maximum file size in bytes. By default, `maxSizeBytes` is -1, which means that the file size is not limited.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE - AMR recording is only supported if AMR is included in the VoiceEngine build. See `Recording and Conversion Formats` for more details.

Example Code

```
GIPS_CodecInst format;

// acquire sub-API (assuming GIPSVoiceEngine object exists)
GIPSVEFile* file = GIPSVEFile::GetInterface(ve);

// record playout on channel 0 to a 10 seconds long WAV-file at 16kHz
format.plfreq = 16000; strcpy(format.plname, "L16");
file->GIPSVE_StartRecordingPlayout(0, "RecordedPlayout16kHz.wav", &format);
SLEEP(10000);
file->GIPSVE_StopRecordingPlayout(0);

// release sub-API
file->Release();
```

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StopRecordingPlayout

This function stops the recording of playout for a specified channel.

Syntax

```
int GIPSVE_StopRecordingPlayout(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard

Header	Declared in GIPSVEFile.h

GIPSVE_StartRecordingMicrophone

This function starts the recording of the microphone signal. Use one of the following two interfaces to record to a file or stream respectively.

Syntax

```
int GIPSVE_StartRecordingMicrophone(const char* fileNameUTF8, GIPS_CodecInst*
   compression = NULL, int channel = -1, int maxSizeBytes = -1);

int GIPSVE_StartRecordingMicrophone(OutStream* stream, GIPS_CodecInst*
   compression = NULL, int channel = -1);
```

Parameters

fileNameUTF8 [in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.

stream [in] A pointer to an `OutStream` derived instance.

compression [in_opt] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for recording.

channel [in_opt] Setting this parameter to another value than default (-1) enables recording of the microphone signal and a locally added file on the specified channel. As an example, if `channel` is set to 0, the file recording will contain a mix of the microphone signal and any added local file on `channel` 0. See `GIPSVE_StartPlayingFileAsMicrophone()` for details on how to add a file to the microphone. It is only possible to enable this type of recording for one channel at a time.

maxSizeBytes [in_opt] The maximum file size in bytes. By default, `maxSizeBytes` is -1, which means that the file size is not limited.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE - AMR recording is only supported if AMR is included in the VoiceEngine build. See `Recording and Conversion Formats` for more details.

Example Code

See `GIPSVE_StartRecordingPlayout()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard

Header	Declared in GIPSVEFile.h

GIPSVE_StopRecordingMicrophone

This function stops the recording of the microphone signal.

Syntax

```
int GIPSVE_StopRecordingMicrophone();
```

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StartRecordingCall

This function starts the recording of the entire call (the mixed playout and microphone signals). Use one of the following two interfaces to record to a file or stream respectively.

Syntax

```
int GIPSVE_StartRecordingCall(const char* fileNameUTF8, GIPS_CodecInst*
   compression = NULL, int maxSizeBytes = -1);

int GIPSVE_StartRecordingCall(OutStream* stream, GIPS_CodecInst* compression =
   NULL);
```

Parameters

fileNameUTF8 [in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.

stream [in] A pointer to an `OutStream` derived instance.

compression [in_opt] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for recording.

maxSizeBytes [in_opt] The maximum file size in bytes. By default, `maxSizeBytes` is -1, which means that the file size is not limited.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE - AMR recording is only supported if AMR is included in the VoiceEngine build. See `Recording and Conversion Formats` for more details.

Example Code

See `GIPSVE_StartRecordingPlayout()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StopRecordingCall

This function stops the recording of the entire call (the mixed playout and microphone signals).

Syntax

```
int GIPSVE_StopRecordingCall();
```

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_PauseRecordingCall

This function pauses the recording of the call (the mixed playout and microphone signals).

Syntax

```
int GIPSVE_PauseRecordingCall(bool enable);
```

Parameters

enable [in] Pause the recording if set to `true`. Continue recording (un-pause) if set to `false`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StartRecordingPlayoutStereo

This function starts the recording of playout in stereo. Use one of the following two interfaces to record to a file or stream respectively.

Syntax

```
GIPSVE_StartRecordingPlayoutStereo(const char* fileNameLeftUTF8, const char*
   fileNameRightUTF8, GIPS_StereoChannel select, GIPS_CodecInst* compression =
   NULL);

int GIPSVE_StartRecordingPlayoutStereo(OutStream* streamLeft, OutStream*
   streamRight, GIPS_StereoChannel select, GIPS_CodecInst* compression = NULL);
```

Parameters

filenameLeftUTF8	[in] A pointer to an array containing the filename (as a UTF-8 encoded null-terminated string) to which the left playout will be recorded.
filenameRightUTF8	[in] A pointer to an array containing the filename (as a UTF-8 encoded null-terminated string) to which the right playout will be recorded.
streamLeft	[in] A pointer to an `OutStream` derived instance to handle the left playout.
streamRight	[in] A pointer to an `OutStream` derived instance to handle the right playout.
select	[in] A `GIPS_StereoChannel` enumerator which specifies what stereo mode to use for recording.
compression	[in_opt] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for recording.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE - AMR recording is only supported if AMR is included in the VoiceEngine build. See `Recording and Conversion Formats` for more details.

The recording takes place just before the signal is played out to the speaker. The mono signals for each channel are first panned to stereo left or right and then mixed. Following this we record the left and right channels.

Requirements

Supported platforms	Windows, MAC OS X
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StopRecordingPlayoutStereo

This function stops the recording of playout in stereo.

Syntax

```
int GIPSVE_StopRecordingPlayoutStereo(GIPS_StereoChannel select);
```

Parameters

select [in] A `GIPS_StereoChannel` enumerator which selects what channel (left, right or both) to stop recording.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ConvertWAVToPCM

These interfaces can be used to convert a WAV file containing data of arbitrary PCM format to a PCM file with 16 kHz sample rate and 16 bits/sample. The converted file can be used when calling `GIPSVE_StartPlayingFileLocally()`. Use one of the following two interfaces to convert a file or stream respectively.

Syntax

```
virtual int GIPSVE_ConvertWAVToPCM(const char* fileNameInUTF8, const char*
  fileNameOutUTF8) = 0;

int GIPSVE_ConvertWAVToPCM(InStream* streamIn, OutStream* streamOut);
```

Parameters

fileNameInUTF8 [in] A pointer to an array containing the WAV filename as a UTF-8 encoded null-terminated string.

fileNameOutUTF8 [in] A pointer to an array containing the PCM filename as a UTF-8 encoded null-terminated string.

streamIn [in] A pointer to an `InStream` derived instance to handle the WAV input.

streamOut [in] A pointer to an `OutStream` derived instance to handle the PCM output.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ConvertPCMToWAV

This function converts a PCM file (16 kHz sample rate and 16 bits/sample) to WAV format. Use one of the following two interfaces to convert a file or stream respectively.

Syntax

```
int GIPSVE_ConvertPCMToWAV(const char* fileNameInUTF8, const char*
   fileNameOutUTF8);

int GIPSVE_ConvertPCMToWAV(InStream* streamIn, OutStream* streamOut, int
   lenghtInBytes);
```

Parameters

fileNameInUTF8 [in] A pointer to an array containing the PCM filename as a UTF-8 encoded null-terminated string.

fileNameOutUTF8 [in] A pointer to an array containing the WAV filename as a UTF-8 encoded null-terminated string.

streamIn [in] A pointer to an `InStream` derived instance to handle the PCM input.

streamOut [in] A pointer to an `OutStream` derived instance to handle the WAV output.

lengthInBytes [in] Length of the input file in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ConvertPCMToCompressed

This function compresses a PCM file (16 kHz sample rate and 16 bits/sample) to a compressed format. Use one of the following two interfaces to convert a file or stream respectively.

Syntax

```
int GIPSVE_ConvertPCMToCompressed(const char* fileNameInUTF8, const char*
   fileNameOutUTF8, GIPS_CodecInst* compression);

int GIPSVE_ConvertPCMToCompressed(InStream* streamIn, OutStream*
   streamOut,GIPS_CodecInst* compression);
```

Parameters

fileNameInUTF8 [in] A pointer to an array containing the PCM filename as a UTF-8 encoded null-terminated string.

fileNameOutUTF8 [in] A pointer to an array containing the compressed filename as a UTF-8 encoded null-terminated string.

streamIn [in] A pointer to an `InStream` derived instance to handle the PCM input.

streamOut [in] A pointer to an `OutStream` derived instance to handle the compressed output.

compression [in] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for conversion.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE - AMR conversion is only supported if AMR is included in the VoiceEngine build. See `Recording and Conversion Formats` for more details.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ConvertCompressedToPCM

This function decompresses a compressed file to PCM format (16 kHz sample rate and 16 bits/sample).

Syntax

```
int GIPSVE_ConvertCompressedToPCM(const char* fileNameInUTF8, const char*
   fileNameOutUTF8);

int GIPSVE_ConvertCompressedToPCM(InStream* streamIn, OutStream* streamOut);
```

Parameters

fileNameInUTF8 [in] A pointer to an array containing the compressed filename as a UTF-8 encoded null-terminated string.

fileNameOutUTF8	[in] A pointer to an array containing the PCM filename as a UTF-8 encoded null-terminated string.
streamIn	[in] A pointer to an `InStream` derived instance to handle the compressed input.
streamOut	[in] A pointer to an `OutStream` derived instance to handle the PCM output.
compression	[in] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for conversion.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

NOTE - AMR conversion is only supported if AMR is included in the VoiceEngine build. See `Recording and Conversion Formats` for more details.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_InitRTPToFileConversion

This function initiates RTP-to-file conversion. Use one of the following two interfaces to record to a file or stream respectively.

Syntax

```
int GIPSVE_InitRTPToFileConversion(const char* fileNameUTF8, unsigned int
   conversionDelay, GIPS_CodecInst* compression = NULL);

int GIPSVE_InitRTPToFileConversion(OutStream* stream, unsigned int
   conversionDelay, GIPS_CodecInst* compression = NULL);
```

Parameters

fileNameUTF8	[in] A pointer to an array containing the output filename as a UTF-8 encoded null-terminated string.
stream	[in] A pointer to an `OutStream` derived instance
conversionDelay	[in] The number of milliseconds that the VoiceEngine delays the mixing of all channels to ensure all packets are received before mixing and storing files.
compression	[in_opt] A pointer to the `GIPS_CodecInst` structure holding the codec information to use for conversion.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Initialize the conversion with `GIPSVE_InitRTPToFileConversion()`, indicate which channels are to be mixed with `GIPSVE_StartRTPToFileConversion()` and finally convert on a packet-by-packet basis with `GIPSVE_ConvertRTPToFile()`.

Note that you need the timestamp for each packet to reproduce the call.

You can choose to store the output to a file or to an `OutStream`. See `Recording and Conversion Formats` for a complete description of the file formats allowed by parameter `compression`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StartRTPToFileConversion

This function allows RTP-to-file conversion to begin for a specific channel.

Syntax

```
int GIPSVE_StartRTPToFileConversion(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_StopRTPToFileConversion

This function stops RTP-to-file conversion to begin for a specific channel.

Syntax

```
int GIPSVE_StopRTPToFileConversion(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_ConvertRTPToFile

This function converts a single RTP packet.

Syntax

```
int GIPSVE_ConvertRTPToFile(int channel, char* rtpPacketBuffer, unsigned int
   length, unsigned int incomingTimeStamp);
```

Parameters

channel [in] The channel ID number.

rtpPacketBuffer [in] A pointer to an array containing the stored RTP packet.

length [in] The length of the RTP packet in bytes.

incomingTimeStamp [in] The time in milliseconds since the beginning of the call when the packet was originally received.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

To flush the internal buffers in VoiceEngine (the conversionDelay), call this function with `rtpPacketBuffer` and `length` set to `0`, and `incomingTimeStamp` set to the last timestamp.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_GetPlaybackPosition

This function returns the current played position of a file on a specific channel.

Syntax

```
int GIPSVE_GetPlaybackPosition(int channel, int& positionMs);
```

Parameters

channel [in] The channel ID number.

positionMs [out] The current played position of the file (in milliseconds).

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVE_GetFileDuration

This function returns the duration of a specified file.

Syntax

```
int GIPSVE_GetFileDuration(const char* fileNameUTF8, int& durationMs,
  GIPS_FileFormats format = FILE_PCM_16KHZ);
```

Parameters

fileNameUTF8 [in] A pointer to an array containing the name of the file as a UTF-8 encoded null-terminated string.

durationMs [out] The duration of the file (in milliseconds).

format [in_opt] A `GIPS_FileFormats` enumerator that specifies the file type.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEFile.h

GIPSVEG729Extended

The `GIPSVEG729Extended` sub-API mainly adds the following functionality to `GIPSVEBase`:

- G.729 Annex-B control.

NOTE 1: GIPSVEG729Extended support requires a special VoiceEngine build configuration.
NOTE 2: The GIPSVEG729Extended:: prefix is excluded for most API names throughout this chapter.

GetInterface

Retrieves a pointer to the GIPSVEG729Extended sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEG729Extended* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created GIPSVoiceEngine object.

Return Values

If the function succeeds, the return value is a pointer to the new GIPSVEG729Extended interface.

If the function fails, the return value is NULL.

Remarks

Each call to this function increments an internal reference counter for the specified GIPSVoiceEngine object. This reference count is decreased by calling the corresponding Release() method and it must be zero when the VoiceEngine instance is deleted (see also GIPSVoiceEngine::Delete()).

Example Code

See GIPSVEBase::GIPSVE_GetInterface().

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEG729Extended.h

Release

Releases the GIPSVEG729Extended sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVEBase::GIPSVE_LastError().

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()`can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEG729Extended.h

GIPSVE_SetG729AnnexBStatus

This function enables, or disables, G.729 Annex-B (VAD/DTX/CNG scheme) for a specific channel.

Syntax

```
int GIPSVE_SetG729AnnexBStatus(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] `false`: Disable G.729 Annex-B VAD/DTX/CNG scheme.
`true`: Enable G.729 Annex-B VAD/DTX/CNG scheme. This is the default setting.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This API only has an effect when the G.729A codec is used as encoder.

G.729 Annex-B is enabled by default.

If G.729 Annex-B is enabled, G.729 Annex-B CNG frames (2 bytes) are included in the G.729A stream (using the same RTP payload type as the G.729A packets) and transmitted during silence periods.

If G.729 Annex-B is disabled, GIPS built in VAD/DTX/CNG scheme (same as for G.711, iLBC, iSAC etc.) is used. CNG packets are then transmitted with a different RTP payload type (default is 13) than the G.729A packets.

Requirements

Supported platforms	Windows, MAC OS X, Linux

VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEG729Extended.h

GIPSVE_GetG729AnnexBStatus

Retrieves the current G.729 Annex-B setting for a specific channel.

Syntax

```
int GIPSVE_GetG729AnnexBStatus(int channel, bool& enabled);
```

Parameters

channel [in] The channel ID number.

enabled [out] A binary reference output which is set to `true` if G.729 Annex-B is enabled and `false` otherwise.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

G.729 Annex-B is enabled by default.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEG729Extended.h

GIPSVE_SetG729SilenceThreshold

THIS FUNCTION IS OBSOLETE AND WILL BE REMOVED IN THE NEXT VoiceEngine VERSION

Syntax

```
int GIPSVE_SetG729SilenceThreshold(int channel, short silenceThreshold);
```

Parameters

channel [in] The channel ID number.

silenceThreshold [in] Refer to GIPS for details.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEG729Extended.h

GIPSVE_SetG729SidResendThreshold

Refer to GIPS for more details regarding this function.

Syntax

```
int GIPSVE_SetG729SidResendThreshold(int channel, short sidThreshold);
```

Parameters

channel [in] The channel ID number.

sidThreshold [in] Refer to GIPS for details.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEG729Extended.h

GIPSVEEncryption

The `GIPSVEEncryption` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Secure RTP (SRTP).
- External encryption and decryption.

NOTE: The `GIPSVEEncryption`:: prefix is excluded for most API names throughout this chapter.

Secure RTP

NOTE: This is an optional feature. It relies on a GIPS product that may not be included in your VoiceEngine configuration.

VoiceEngine can be delivered with a reference implementation of Secure RTP (SRTP) using the open source libSRTP available at http://srtp.sourceforge.net/srtp.html. A brief description of the functionality is given here; for complete information please refer to the libSRTP webpage, RFC 3711 and http://en.wikipedia.org/wiki/Secure_Real-time_Transport_Protocol. Please also refer to the SRTP API calls below.

There are two types of protection, encryption and authentication. Both, one or none of them can be used, creating four combined types of protection. It is strongly recommended to use both. Encryption is done on the payload; authentication is applied to header and payload.

There are two kinds of ciphers (encryption algorithms) available, AES 128 counter mode and null. There are two kinds of authentication available, HMAC-SHA1 and null. Null cipher and null authentication provide no protection and are available for compliance with RFC 3711.

A session encryption key, a session authentication key and a session salt key are derived from a master key and master salt. The master key is assumed to be 128 bits (16 bytes) and the master salt is assumed to be 112 bits (14 bytes) for the supported cipher and authentication types. The key input parameter to the API calls is a pointer to a buffer that contains both the master key (first 128 bits) and master salt (the following 112 bits) and is consequently assumed to be 240 bits (30 bytes). For information on master key handling, please refer to the libSRTP webpage.

For AES 128 CM, the cipher key length is the session encryption key length plus session salt key length. The session encryption key length is always 128 bits (16 bytes). For null cipher, the cipher key length is the session encryption key length.

The authentication key length is the session authentication key length.

Authentication tag length is the length of the authentication tag added to the packet.

Recommended parameters

Desired protection	Both (recommended)	Encryption only	Authentication only
Cipher type	AES 128 CM	AES 128 CM	NULL
Cipher length	30	30	0
Authentication type	HMAC-SHA1	NULL	HMAC-SHA1
Authentication key length	20	0	20
Authentication tag length	4 or 10[1]	0	4 or 10[1]
Security level	Encryption and auth	Encryption	Authentication

Valid parameter values

Cipher length	16 - 256
Authentication key length	0 – 20 (HMAC-SHA1) / 0 – 256 (NULL)
Authentication tag length	0 – 20 (HMAC-SHA1) / 0 – 12 (NULL)

NOTE: The valid parameters values are only relevant when the corresponding protection is enabled.

Enumerator GIPS_CipherTypes

Syntax

```
enum GIPS_CipherTypes
```

[1] Authentication tag length of 4 is recommended for voice only. Length 10 is recommended for other applications, such as DTMF. If DTMF security is needed during a voice call, length 10 is recommended.

```
{
    CIPHER_NULL = 0,
    CIPHER_AES_128_COUNTER_MODE
};
```

Enumerator GIPS_AuthenticationTypes

Syntax

```
enum GIPS_AuthenticationTypes
{
    AUTH_NULL = 0,
    AUTH_HMAC_SHA1 = 3
};
```

Enumerator GIPS_SecurityLevels

Syntax

```
enum GIPS_SecurityLevels
{
    NO_PROTECTION = 0,
    ENCRYPTION,
    AUTHENTICATION,
    ENCRYPTION_AND_AUTHENTICATION
};
```

GetInterface

Retrieves a pointer to the `GIPSVEEncryption` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEEncryption* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEEncryption` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

Release

Releases the `GIPSVEEncryption` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVE_EnableSRTPSend

This function enables SRTP on the transmitted data for a specific channel.

Syntax

```
int GIPSVE_EnableSRTPSend(int channel, GIPS_CipherTypes cipherType, unsigned int
   cipherKeyLength, GIPS_AuthenticationTypes authType, unsigned int
   authKeyLength, unsigned int authTagLength, GIPS_SecurityLevels level, const
   unsigned char* key, bool useForRTCP = false);
```

Parameters

channel [in] The channel ID number.

cipherType [in] Cipher type.

cipherKeyLength [in] Cipher key length.

authType [in] Authentication type.

authKeyLength [in] Authentication key length.

authTagLength [in] Authentication tag length.

level [in] Security type.

key [in] Pointer to the buffer containing the master key and master salt. This buffer must always be 30 bytes. See the `Secure RTP` description above.

useForRTCP [in] Set to true to enable protection also for RTCP packets. If set to false, only RTP packets are protected.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVE_DisableSRTPSend

This function disables SRTP on the transmitted data for a specific channel.

Syntax

```
int GIPSVE_DisableSRTPSend(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVE_EnableSRTPReceive

This function enables SRTP on the received data for a specific channel.

Syntax

```
int GIPSVE_EnableSRTPReceive(int channel, GIPS_CipherTypes cipherType, unsigned
  int cipherKeyLength, GIPS_AuthenticationTypes authType, unsigned int
  authKeyLength, unsigned int authTagLength, GIPS_SecurityLevels level, const
  unsigned char* key, bool useForRTCP = false);
```

Parameters

channel	[in] The channel ID number.
cipherType	[in] Cipher type.
cipherKeyLength	[in] Cipher key length.
authType	[in] Authentication type.
authKeyLength	[in] Authentication key length.
authTagLength	[in] Authentication tag length.
level	[in] Security type.
key	[in] Pointer to the buffer containing the master key and master salt. This buffer must always be 30 bytes. See the `Secure RTP` description above.
useForRTCP	[in] Set to true to enable protection also for RTCP packets. If set to false, only RTP packets are protected.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVE_DisableSRTPReceive

This function disables SRTP on the received data for a specific channel.

Syntax

```
int GIPSVE_DisableSRTPReceive(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVE_InitEncryption

This function installs a `GIPS_encryption` derived instance.

Syntax

```
int GIPSVE_InitEncryption(GIPS_encryption* encryptionObject);
```

Parameters

encryptionObject [in] A pointer to the derived instance. NULL is an acceptable value that will uninstall a previously installed object, and also disable encryption for all channels.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVE_SetEncryptionStatus

This function enables, or disables, encryption and decryption for a specific channel.

Syntax

```
GIPSVE_SetEncryptionStatus(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] Enables encryption and decryption if set to `true`. Disables encryption and decryption if set to `false`.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux, Symbian (S60 3rd Ed.), iPhone, Android
VE configuration	Standard
Header	Declared in GIPSVEEncryption.h

GIPSVEPTT

The `GIPSVEPTT` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- The Push-to-Talk (PTT) enables play out only as incoming packets are received.
- When remote talkers are silent, play out is stopped.

NOTE: The `GIPSVEPTT::` prefix is excluded for most API names throughout this chapter.

GetInterface

Retrieves a pointer to the `GIPSVEPTT` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEPTT* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEPTT` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEPTT.h

Release

Releases the GIPSVEPTT sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEPTT.h

GIPSVE_StartPTTPlayout

This method enables Push-to-Talk for a specific channel.

Syntax

```
int GIPSVE_StartPTTPlayout(int channel);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The Push-to-Talk (PTT) feature enables play out only as incoming packets are received. When remote talkers are silent, play out is stopped. No calls to `GIPSVE_StartPlayout()` or `GIPSVE_StopPlayout()` are required when using this feature. No mixing takes place: each participant can only hear one other participant at a time.

Use `GIPSVE_StopPlayout()` to stop PTT.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEPTT.h

GIPSVE_GetPTTActivity

This function returns whether PTT activity is currently ongoing for a specific channel.

Syntax

```
int GIPSVE_GetPTTActivity(int channel, bool& activity);
```

Parameters

channel [in] The channel ID number.

activity [out] Set to `true` at return if PTT is active. Set to`false` if PTT is inactive.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard

Header	Declared in GIPSVEPTT.h

GIPSVEVideoSync

The `GIPSVEVideoSync` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- RTP header modification (time stamp and sequence number fields).
- Playout delay tuning to synchronize the voice with video.
- Playout delay monitoring.

NOTE: The `GIPSVEVideoSync::` prefix is excluded for most API names throughout this chapter.

GetInterface

Retrieves a pointer to the `GIPSVEVideoSync` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEVideoSync* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEVideoSync` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

Release

Releases the `GIPSVEVideoSync` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVE_GetPlayoutTimeStamp

This function returns the RTP timestamp of the audio that is currently being played out.

Syntax

```
int GIPSVE_GetPlayoutTimestamp(int channel, unsigned int& timestamp);
```

Parameters

channel [in] The channel ID number.

timestamp [out] The RTP timestamp.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks
This function can be used to synchronize video with the voice stream.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVE_SetInitTimestamp
This function allows manual initialization of the RTP timestamp.

Syntax

```
int GIPSVE_SetInitTimestamp(int channel, unsigned int timestamp);
```

Parameters
channel [in] The channel ID number.

timestamp [in] The RTP timestamp.

Return Values
The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks
If this call has been made the channel must be re-created for VoiceEngine to initialize the RTP timestamp.

This call should be performed before `GIPSVEBase::GIPSVE_StartSend()` is called.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVE_SetInitSequenceNumber
This function allows manual initialization of the RTP sequence number.

Syntax

```
int GIPSVE_SetInitSequenceNumber(int channel, short sequenceNumber);
```

Parameters
channel [in] The channel ID number.

sequenceNumber [in] The RTP sequence number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

If this call has been made the channel must be re-created for VoiceEngine to initialize the RTP sequence number.

This call should be performed before `GIPSVEBase::GIPSVE_StartSend()` is called.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVE_SetMinimumPlayoutDelay

This function sets an additional delay for the playout jitter buffer.

Syntax

```
int GIPSVE_SetMinimumPlayoutDelay(int channel, int delayMs);
```

Parameters

channel [in] The channel ID number.

delayMs [in] The additional playout delay, in milliseconds. Allowed range is 0-1000 ms.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This function increases the playout delay by the specified amount. By default it is zero. This enables you to synchronize the voice with video.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVE_GetDelayEstimate

This function returns the sum of the algorithmic delay, jitter buffer delay, and the playout buffer delay for a channel.

Syntax

```
int GIPSVE_GetDelayEstimate(int channel, int& delayMs);
```

Parameters

channel	[in] The channel ID number.
delayMs	[out] The delay estimate in milliseconds.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This value does not include the transmission delay.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVE_GetSoundcardBufferSize

This function returns the current sound card buffer size (playout delay).

Syntax

```
int GIPSVE_GetSoundcardBufferSize(int& bufferMs);
```

Parameters

bufferMs	[out] The current sound card buffer size in milliseconds.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This is the delay that occurs between the decoding of a packet and its playout.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEVideoSync.h

GIPSVEVQMon

The GIPSVEVQMon sub-API mainly adds the following functionalities to GIPSVEBase:

- Call quality estimates based on Telchemy VQMon.
- RTCP extended reports (RTCP-XR).
- VQMon callback notifications for predefined conditions.
- VoIP metric reports according to RFC3611.
- SIP service quality reports.

NOTE: The GIPSVEVQMon:: prefix is excluded for most API names throughout this chapter.

Class GIPSVEVQMonCallback

This is a callback class for VQMon alerts.

The VoiceEngine user should override the OnVQMonAlert() method in a derived class. OnVQMonAlert() will be called when VQMon throws an alert.

NOTE: This is an optional feature. It relies on a GIPS product that may not be included in your VoiceEngine configuration.

Syntax

```
class GIPSVEVQMonCallback
{
public:
    virtual OnVQMonAlert(unsigned int instance, int channel, void *alertDesc) =
   0;
};
```

Parameters

instance	[in] The VoiceEngine instance number, numbered sequentially from 1 in the order of creation. (Same number as default trace file name.)
channel	[in] The channel ID number.
alertDesc	[out] Pointer to structure defined by the Telchemy VQMon API. Refer to Telchemy VQMon documentation for details.

Remarks

The derived class is installed with GIPSVE_SetVQMonAlertCallback().

GetInterface

Retrieves a pointer to the GIPSVEVQMon sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEVQMon* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEVQMon` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

Release

Releases the `GIPSVEVQMon` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

Telchemy VQMon Support

This section lists calls that are available to control Telchemy VQMon. VQMon monitors voice calls and produces call quality estimates. The VoiceEngine can be integrated with VQMon-EP into a simple high-level API. The VQMon software needs to be licensed from Telchemy for this functionality. Refer to http://www.telchemy.com/vqmonep.html for more information.

The total number of channels that can run VQMon is limited and independent of the number of VoiceEngine instances. These available channels can be arbitrarily spread over the instances.

NOTE: VQMon support requires a special VoiceEngine build configuration.

GIPSVE_SetVQMonStatus

This function enables, or disables, VQMon for a specific channel.

Syntax

```
int GIPSVE_SetVQMonStatus(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] `false`: Disable VQMon.
`true`: Enable VQMon.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

This will return an error if the maximum number of VQMon channels allowed is reached.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVE_SetRTCPXRStatus

This function enables, or disables, RTCP extended reports (RTCP-XR) for a specific channel. For more information see RFC 3611.

Syntax

```
int GIPSVE_SetRTCPXRStatus(int channel, bool enable);
```

Parameters

channel [in] The channel ID number.

enable [in] `false`: disable RTCP-XR
`true`: enable RTCP-XR

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

RTCP must first be enabled for this channel with `GIPSVE_SetRTCPStatus()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVE_SetVQMonAlertCallback

This function installs a VQMon alert handler callback function. The function will be called for any VQMon alert.

Syntax

```
int GIPSVE_SetVQMonAlertCallback(GIPSVEVQMonCallback *vqmonCallback);
```

Parameters

vqmonCallback [in] Pointer to an instance of a `GIPSVEVQMonCallback` derived class.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h and external Telchemy VQMon header files.

GIPSVE_EnableVQMonAlert

This function sets an alert type for a certain condition to a specific channel. Refer to Telchemy VQMon documentation for parameter descriptions.

Syntax

```
int GIPSVE_EnableVQMonAlert(int channel, int type, int param1[4], int param2[4],
   int param3[4]);
```

Parameters

channel [in] The channel ID number.

type [in] Alert type.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVE_DisableVQMonAlert

This function removes an alert type from a specific channel.

Syntax

```
int GIPSVE_DisableVQMonAlert(int channel, int type);
```

Parameters

channel [in] The channel ID number.

type [in] Alert type.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVE_GetVoipMetrics

This function returns a VoIP Metrics Report Block for the specified channel. For more information see RFC 3611, section 4.7.

Syntax

```
GIPSVE_GetVoipMetrics(int channel, unsigned char* data, unsigned int& length);
```

Parameters

channel [in] The channel ID number.

data [out] A pointer to an array to which the VoIP Metrics block will be copied.

length [out] The size of the array pointed to by `data` in bytes.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVE_GetVQMonSIPReport

This function generates a SIP service quality report. Refer to Telchemy VQMon documentation for parameter descriptions.

Syntax

```
int GIPSVE_GetVQMonSIPReport(int channel, unsigned char* data, unsigned int&
   length, char strSIPLocalCallID[80], char strSIPRemoteCallID[80], char
   strSIPLocalStartTimestamp[30], char strSIPLocalStopTimestamp[30], char
   strSIPRemoteStartTimestamp[30], char strSIPRemoteStopTimestamp[30]);
```

Parameters

channel [in] The channel ID number.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVE_VQMonIPInfo

This function provides VQMon with the call setup information to include in the SIP report. This is normally used with external transport, as VoiceEngine would not have this information.

Syntax

```
int GIPSVE_SetVQMonIPInfo(int channel, const unsigned char* localIP, int
   localPort, const unsigned char* remoteIP, int remotePort);
```

Parameters

channel	[in] The channel ID number.
localIP	[in] A pointer to an array containing the local IP address as a null-terminated string.
localPort	[in] The local receive port.
remoteIP	[in] A pointer to an array containing the remote IP address as a null-terminated string.
remotePort	[in] The remote receive port.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Requires a special VoiceEngine build configuration
Header	Declared in GIPSVEVQMon.h

GIPSVEExternalMedia

In some cases it is desirable to use an audio source or sink which may not be available to the VE, such as a DV camera. This section lists functions that allow for the use of such external recording sources and playout sinks. It also describes how recorded data, or data to be played out, can be modified outside the VE.

NOTE: The `GIPSVEExternalMedia`:: prefix is excluded for most API names throughout this chapter.

Enumerator GIPS_ProcessingTypes

This enumerator is used to specify where - in the audio path - to install the external media processing. It is utilized by the `GIPSVE_SetExternalMediaProcessing()` API.

Syntax

```
enum GIPS_ProcessingTypes
```

```
{
    PLAYBACK_PER_CHANNEL = 0,
    PLAYBACK_ALL_CHANNELS_MIXED,
    RECORDING_PER_CHANNEL,
    RECORDING_ALL_CHANNELS_MIXED
};
```

Enumerators

PLAYBACK_PER_CHANNEL	The received audio for one channel, before it is played out.
PLAYBACK_ALL_CHANNELS_MIXED	The mixed audio that is played out, including all channels and files. The channel argument is not relevant for this case.
RECORDING_PER_CHANNEL	The microphone signal for one specific channel, following GIPS voice processing.
RECORDING_ALL_CHANNELS_MIXED	The mixed audio for all recording channels. The channel argument is not relevant for this case.

Class GIPSVEMediaProcess

This is a callback class for processing audio externally.

The VoiceEngine user should override the `Process()` method in a derived class. `Process()` will be called when audio is ready to be processed. The audio can be accessed in several different places.

Syntax

```
class GIPSVEMediaProcess
{
public:
    void Process(int channelNumber, short* audioLeft10ms, short* audioRight10ms,
   int length, int  samplingFreq, bool isStereo) = 0;
};
```

Parameters

channelNumber	[in] The channel ID number.
audioLeft10ms	[in] Pointer to an array containing a 10 ms frame of audio for the left channel.
audioRight10ms	[in] Pointer to an array containing a 10 ms frame of audio for the right channel. Will be NULL if *isStereo* is false.
length	[in] The length of the array pointed to by `audio10ms16kHz` in bytes.
samplingFreq	[in] The audio sampling frequency.
isStereo	[in] If true, *audioRight10ms* will point to an array containing audio for the right channel.

Remarks

The derived class is installed with `GIPSVE_SetExternalMediaProcessing()`.

`Process()` will be called in 10 ms intervals. The function should modify the original data and ensure it is copied back to the `audioLeft10ms and audioRight10ms` arrays. The number of samples in the frame cannot be changed. The sampling frequency will depend upon the codec used.

Both the left and right stereo channels need to be processed if VoiceEngine is playing out in stereo, for example if the audio is panned using `GIPSVE_SetOutputVolumePan()` or `GIPSVE_SetChannelOutputVolumePan()`.Recording in stereo is currently not supported and *isStereo* will always be false.

GetInterface

Retrieves a pointer to the `GIPSVEExternalMedia` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVEExternalMedia* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine [in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVEExternalMedia` interface.

If the function fails, the return value is `NULL`.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

Release

Releases the `GIPSVEExternalMedia` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

GIPSVE_SetExternalMediaProcessing

This function installs a `GIPSVEMediaProcess` derived instance.

Syntax

```
int GIPSVE_SetExternalMediaProcessing(GIPS_ProcessingTypes type, int channel,
   bool enable, GIPSVEMediaProcess& proccessObject);
```

Parameters

type [in] A `GIPS_ProcessingTypes` enumerator which specifies the location where the audio should be accessed.

channel [in] The channel ID number.

enable [in] `false`: Disable the callback.
`true`: Enable the callback.

proccessObject [in] The `GIPSVEMediaProcess` derived instance.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Requirements

Supported platforms	Windows (incl. CE/Mobile), MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

GIPSVE_SetExternalRecording

This function enables external recording.

Syntax

```
int GIPSVE_SetExternalRecording(bool enable);
```

Parameters

enable [in] `false`: disable external recording
`true`: enable external recording

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

External recording must be enabled before transmission is started for any channel, otherwise a standard sound device will be used as the recording source.

External recording cannot be disabled as long as a channel is sending.

When enabled, `GIPSVE_ExternalRecordingInsertData()` must be used to pass in the externally recorded audio.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

GIPSVE_SetExternalPlayout

This function enables external playout.

Syntax

```
int GIPSVE_SetExternalPlayout(bool enable);
```

Parameters

enable [in] `false`: disable external playout
`true`: enable external playout

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

External playout must be enabled before transmission is started for any channel, otherwise a standard sound device will be used as the playout sink.

External playout cannot be disabled as long as a channel is sending.

When enabled, `GIPSVE_ExternalPlayoutGetData()` must be used to receive the audio for external playout.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

GIPSVE_ExternalRecordingInsertData

This function accepts externally recorded audio.

Syntax

```
GIPSVE_ExternalRecordingInsertData(const short* speechData10ms, unsigned int
   lengthSamples, int samplingFreqHz, int currentDelayMs);
```

Parameters

speechData10ms [in] A pointer to an array containing a frame of audio.

lengthSamples [in] The length of the audio frame in samples, which must be a multiple of 10 milliseconds. The frame length must thus be a multiple of 160 or 480 (for 16 or 48 kHz sampling rates respectively).

samplingFreqHz [in] The sampling frequency of the audio, in Hz (16000 or 48000) .

currentDelayiMs [in] An estimate of the delay (in milliseconds) from the time that the audio is recorded until it is handed to the VoiceEngine. This estimate is important to the echo canceller

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

During transmission, this function should be called at as regular an interval as possible with frames of corresponding size.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

GIPSVE_ExternalPlayoutGetData

This function gets audio for an external playout sink.

Syntax

```
GIPSVE_ExternalPlayoutGetData(short* speechData10ms, int samplingFreqHz, int
   currentDelayMs, unsigned int& lengthSamples);
```

Parameters

speechData10ms [in] A pointer to an array to which a 10 ms frame of audio will be copied.

samplingFreqHz [in] The sampling frequency desired for playback, in Hz (16000 or 48000).

currentDelayMs [in] An estimate of the delay (in milliseconds) from the time of receiving the audio until it is played out. This estimate is important to the echo canceller.

lengthSamples [out] Will contain the length of the audio frame in samples at return.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

During transmission, this function should be called every 10 ms to obtain a new 10 ms frame of audio. The length of the block will be either 160 or 480 samples (for 16 or 48 kHz sampling rates respectively).

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVEExternalMedia.h

GIPSVECallReport

The `GIPSVECallReport` sub-API mainly adds the following functionalities to `GIPSVEBase`:

- Long-term speech and noise level metrics.
- Long-term echo metric statistics.
- Round Trip Time (RTT) statistics.

- Dead-or-Alive connection summary.
- Generation of call reports to text file.

NOTE: The `GIPSVECallReport::` prefix is excluded for most API names throughout this chapter.

Struct GIPS_stat_val

Syntax

```
struct GIPS_stat_val
{
    int min;
    int max;
    int average;
};
```

Parameters

min	The minimum value.
max	The maximum value.
average	The average value.

Struct GIPS_P56_statistics

Syntax

```
struct GIPS_P56_statistics
{
   GIPS_stat_val speechRx;
   GIPS_stat_val speechTx;
   GIPS_stat_val noiseRx;
   GIPS_stat_val noiseTx;
};
```

Parameters

speechRx	Long-term speech levels (min, max and average) on the receiving side.
speechTx	Long-term speech levels on the sending side.
noiseRx	Long-term noise/silence levels on the receiving side.
noiseTx	Long-term noise/silence levels on the sending side.

Remarks

All levels are reported in dBm0.

Struct GIPS_echo_statistics

Syntax

```
struct GIPS_echo_statistics
{
  GIPS_stat_val ERL;
  GIPS_stat_val ERLE;
  GIPS_stat_val RERL;
  GIPS_stat_val A_NLP;
};
```

Parameters

ERL	Echo Return Loss metrics (min, max and average).
ERLE	Echo Return Loss Enhancement metrics.
RERL	RERL = ERL + ERLE.
A_NLP	Echo suppression inside the EC at the point just before its NLP

Remarks

All levels are reported in dB.

GetInterface

Retrieves a pointer to the `GIPSVECallReport` sub-API and increases an internal reference counter for this sub API.

Syntax

```
static GIPSVECallReport* GetInterface(GIPSVoiceEngine* voiceEngine);
```

Parameters

voiceEngine	[in] Pointer to an already created `GIPSVoiceEngine` object.

Return Values

If the function succeeds, the return value is a pointer to the new `GIPSVECallReport` interface. If the function fails, the return value is NULL.

Remarks

Each call to this function increments an internal reference counter for the specified `GIPSVoiceEngine` object. This reference count is decreased by calling the corresponding `Release()` method and it must be zero when the VoiceEngine instance is deleted (see also `GIPSVoiceEngine::Delete()`).

Example Code

See `GIPSVEBase::GIPSVE_GetInterface()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

Release

Releases the `GIPSVECallReport` sub-API and decreases an internal reference counter for this sub API.

Syntax

```
int Release();
```

Return Values

If the function succeeds, the return value is the value of the internal reference count, which can be used for diagnostic purposes.

If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The number of calls to `Release()` should always match the number of calls to `GetInterface()`.

When the reference count of all sub-APIs reaches zero, `GIPSVoiceEngine::Delete()` can be performed to release the allocated resources.

It is considered safe to delete the VoiceEngine instance even if `Release()` has been called too many times; however, `-1` is given as return value to indicate that the number of calls to `Release()` does not match the number of calls to `GetInterface()`.

Example Code

See `GIPSVEBase::GIPSVE_Release()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

GIPSVE_ResetCallReportStatistics

Performs a combined reset of all components involved in generating the call report.

Syntax

```
int GIPSVE_ResetCallReportStatistics(int channel);
```

Parameters

channel [in] The channel ID number. If `channel` is set to -1, all active channels are reset.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

The channel parameter only affects the dead-or-alive and round-trip-delay measurements. Speech, noise and echo metrics are not channel dependent.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

GIPSVE_GetSpeechAndNoiseSummary

Retrieves minimum, maximum and average levels for long-term speech and noise metrics.

Syntax

```
int GIPSVE_GetSpeechAndNoiseSummary(GIPS_P56_statistics& stats);
```

Parameters

stats [out] A `GIPS_P56_statistics` structure which is filled with speech and noise metrics (min, max and average) on return.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Metrics must first be enabled using the `GIPSVEVQE::GIPSVE_SetMetricsStatus()` API. If metrics are not enabled, all outputs will be set to -100 [dBm0].

See `GIPSVEVQE::GIPSVE_GetSpeechMetrics()` and `GIPSVEVQE::GIPSVE_GetNoiseMetrics()` for more details on the obtained statistics.

The results are derived based on the combined/mixed signals in both the receiving and the transmitting sides.

Call `GIPSVE_ResetCallReportStatistics()`to reset the statistics.

Example Code

```
GIPS_P56_statistics stats;

// acquire sub-APIs (assuming GIPSVoiceEngine object exists)
GIPSVECallReport* report = GIPSVECallReport::GetInterface(ve);
GIPSVEVQE* vqe = GIPSVEVQE::GetInterface(ve);

// enable metrics and reset the statistics for channel 0
vqe->GIPSVE_SetMetricsStatus(true);
report->GIPSVE_ResetCallReportStatistics(0);

// start full duplex VoIP call on channel 0 and collect speech and noise metrics…

// after N (N>>0) seconds, collect a statistical summary
report->GIPSVE_GetSpeechAndNoiseSummary(stats);

// present and/or store the results (see example output below)
DISPLAY_SPEECH_AND_NOISE_METRICS(stats);

// stop the VoIP call

// release sub-APIs
report->Release();
vqe->Release();
```

Example Output

The example above can generate the following output given a one minute long VoIP session between two GIPS clients:

```
Long-term Speech Levels
```

```
Transmitting (TX) side:

  min:  -48 [dBm0]

  max:  -35 [dBm0]

  avg:  -43 [dBm0]

Receiving (RX) side:

  min:  -22 [dBm0]

  max:  -20 [dBm0]

  avg:  -21 [dBm0]
```

```
Long-term Noise Levels
```

```
Transmitting (TX) side:

  min:  -60 [dBm0]

  max:  -58 [dBm0]

  avg:  -59 [dBm0]

Receiving (RX) side:
```

```
min:  -57 [dBm0]
max:  -55 [dBm0]
avg:  -56 [dBm0]
```

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

GIPSVE_GetEchoMetricSummary

Retrieves minimum, maximum and average levels for long-term echo metrics.

Syntax

```
int GIPSVE_GetEchoMetricSummary(GIPS_echo_statistics& stats);
```

Parameters

stats [out] A `GIPS_echo_statistics` structure which is filled with echo metrics (min, max and average) on return.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Metrics must first be enabled using the `GIPSVEVQE::GIPSVE_SetMetricsStatus()` API. If metrics are not enabled, all outputs will be set to -100 [dB].

The Echo Control (EC) must be enabled while echo metrics are collected to generate valid results. Use `GIPSVEVQE::GIPSVE_SetECStatus()` to enable the EC. If the EC is disabled, all outputs will be set to -100 dB.

See `GIPSVEVQE::GIPSVE_GetEchoMetrics()` for more details on the obtained statistics.

Call `GIPSVE_ResetCallReportStatistics()` to reset the statistics.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

GIPSVE_GetRoundTripTimeSummary

Retrieves minimum, maximum and average levels for Round Trip Time (RTT) measurements.

Syntax

```
int GIPSVE_GetRoundTripTimeSummary(int channel, GIPS_stat_val& delaysMs);
```

Parameters

channel	[in] The channel ID number.
stats	[out] A `GIPS_stat_val` structure which is filled with RTT measurements (min, max and average) on return.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

Since RTT measurements are based on RTCP, RTCP must first be enabled using `GIPSVERTP_RTCP::GIPSVE_SetRTCPStatus()`. If RTCP is disabled, all outputs will be set to -1.

Call `GIPSVE_ResetCallReportStatistics()` to reset the statistics.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

GIPSVE_GetDeadOrAliveSummary

Retrieves total amount of dead and alive connection detections during a VoIP session.

Syntax

```
int GIPSVE_GetDeadOrAliveSummary(int channel, int& numOfDeadDetections, int&
   numOfAliveDetections);
```

Parameters

channel	[in] The channel ID number.
numOfDeadDetections	[out] Total number of “dead connection” detections since last reset.
numOfAliveDetections	[out] Total number of “alive connection” detections since last reset.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

`GIPSVENetwork::GIPSVE_SetPeriodicDeadOrAliveStatus()` must first be called with `true` as input parameter to ensure that dead-or-alive detection is enabled. If it is disabled, all output results will be -1.

Call `GIPSVE_ResetCallReportStatistics()` to reset the statistics.

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

GIPSVE_WriteReportToFile

Creates a text file in ASCII format, which contains a summary of all the statistics that can be obtained by the GIPSVECallReport sub API.

Syntax

```
int GIPSVE_WriteReportToFile(const char* fileNameUTF8);
```

Parameters

fileNameUTF8 [in] Pointer to a zero-terminated and UTF-8 encoded character string which contains the name of output file.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling `GIPSVEBase::GIPSVE_LastError()`.

Remarks

To ensure that the output file only contains valid results, the following functions must be enabled during the measurements:

- A full-duplex VoIP session with at least one active channel in each direction;
- Speech, Noise and Echo metrics (see `GIPSVE_SetMetricsStatus()` and `GIPSVE_SetECStatus()`);
- RTT measurements (see `GIPSVE_SetRTCPStatus()`), and
- Dead-or-alive detections (see `GIPSVE_SetPeriodicDeadOrAliveStatus()`).

Call `GIPSVE_ResetCallReportStatistics()` to reset the statistics.

If this API is called two times using the same file name, the first file is overwritten at the second call.

Different file names must be utilized if a set of unique results is required.

Example Code

```
// acquire required sub-APIs
GIPSVECallReport* report = GIPSVECallReport::GetInterface(ve);
GIPSVEVQE* vqe = GIPSVEVQE::GetInterface(ve);
GIPSVERTP_RTCP rtcp = GIPSVERTP_RTCP:: GetInterface(ve);
GIPSVENetwork netw = GIPSVENetwork:: GetInterface(ve);

// enable all parts needed to create a complete call report for channel 0
vqe->GIPSVE_SetMetricsStatus(true);
vqe->GIPSVE_SetECStatus(true);
rtcp->GIPSVE_SetRTCPStatus(0, true);
```

```
netw->GIPSVE_SetPeriodicDeadOrAliveStatus(0, true);

// reset the call-report statistics
report->GIPSVE_ResetCallReportStatistics(0);

// start full duplex VoIP call on channel 0 and perform measurements during a call…

// after N (N>>0) seconds, create a call report and store it on a text file
report->GIPSVE_WriteReportToFile("call_report_1.txt");

// stop the VoIP call

// release sub-APIs
report->Release();
vqe->Release();
rtcp->Release();
netw->Release();
```

Requirements

Supported platforms	Windows, MAC OS X, Linux
VE configuration	Standard
Header	Declared in GIPSVECallReport.h

6 Example code

This chapter describes an example SIP call setup scenario and what calls that needs to be made to the GIPS VE. The order of the function calls in the example is recommended.

The following example illustrates how the above functions should be used during a SIP call setup. The main program needs to take care of all SIP messages and to call the GIPS VoiceEngine when appropriate. The left column in the table shows what is being done in the main program and the right column shows what calls the main program should make to the GIPS VoiceEngine. The main program starts the call-setup by sending an INVITE and it ends with receiving 200 OK for the sent BYE.

What happens in the main program?	Sub-API	What calls needs to be made to the GIPS VoiceEngine?
The application is started		
Initiation of the VoiceEngine The VoiceEngine has now initiated all codecs, NetEq etc	GIPSVEBase	GIPSVE_Init()
Create a new channel	GIPSVEBase	GIPSVE_CreateChannel()
Start without silence suppression and echo cancellation	GIPSVEVQE	GIPSVE_SetVADStatus(false) GIPSVE_SetECStatus(false)
Want to send an INVITE for a new call. Ask for the preferred and available codecs. Get the preferred one and one more!	GIPSVECodec	GIPSVE_NumOfCodecs() GIPSVE_GetCodec(0, codecinfo0) GIPSVE_GetCodec(1, codecinfo1)
Send the INVITE Set the listening port number to 22002 and start listening	GIPSVEBase	GIPSVE_SetLocalReceiver(0, 22002) GIPSVE_StartListen(0)
Receive 180 Ringing, 200 OK User can start sending traffic	GIPSVEBase	GIPSVE_SetSendDestionation(0, 15080, "143.12.12.13") GIPSVE_SetSendCodec(0, codecinfo0) GIPSVE_StartSend(0)
Send ACK User starts playing out to speakers	GIPSVEBase	GIPSVE_StartPlayout(0)
User increases speaker volume	GIPSVEVolumeControl	GIPSVE_GetSpeakerVolume(vol) GIPSVE_SetSpeakerVolume(vol+10)
Conversation is going on	⇔	
User ends call with BYE Stops transmitting packets	GIPSVEBase	GIPSVE_StopSend(0)
User receives 200 OK for BYE Stops listening for packets Stops playing out on speaker	GIPSVEBase	GIPSVE_StopListen(0) GIPSVE_StopPlayout(0)
The application is stopped		
Delete the channel and terminate all VoiceEngine functions.	GIPSVEBase	GIPSVE_DeleteChannel(0) GIPSVE_Terminate()

7 Customizing Codec Settings

This chapter discusses customizing codecs with certain permitted settings.

After retrieving a `GIPS_CodecInst` structure with `GIPSVECodec::GIPSVE_GetCodec()`, certain parameters can be modified for some codecs. Of the six codec parameters, three can be changed: payload type (`pltype`), packet size (`pacsize`) and rate (`rate`). The payload name (`plname`), sampling frequency (`plfreq`), and number of channels (`channels`) can never be modified.

As an example, the `GIPS_CodecInst` structure for iPCM-wb has the following parameters:

- `pltype` = 97
- `plname` = "iPCMWB"
- `plfreq` = 16000
- `channels` = 1
- `rate` = 80000

The payload type of any codec can be changed to support dynamic payload types. Allowed payload types are all values in the dynamic payload range, 96 – 127, and the usual static payload type if one is assigned to the codec.

The packet size of some codecs can be changed. The following table lists codecs with corresponding allowed packet sizes:

Codec	Allowable Packet Sizes (samples)
IPCM-wb	160, 320, 480 and 640
EG711	80, 160, 240 and 320
PCMU/A	80, 160, 240 and 320
ISAC wideband mode	480 and 960
ISAC super-wideband mode	960
ILBC	240

NOTE: iSAC super-wideband mode uses 32 kHz sampling frequency; iPCM-wb and iSAC wideband mode uses 16 kHz sampling frequency, whereas the others use 8 kHz sampling frequency. 320 samples at 32 kHz, 160 samples at 16 kHz, and 80 samples at 8 kHz all corresponds to 10 ms of audio.

The rate can be changed only with the AMR, iSAC or G.729.1 codecs:

- **AMR** has possible rates of: 4750, 5150, 5900, 6700, 7400, 7950, 10200, and 12200. The rate can also be set to an integer, 0 – 7, for a corresponding bit rate of 4750 – 12200.

- **iSAC** can be used in either an channel-adaptive or instantaneous (static target bitrate) mode. To use a static target bitrate, specify a rate between 10000 and 32000 bits/s if iSAC is setup to run in wideband mode (plfreq = 16000), or between 10000 and 56000 bits/s if iSAC is setup to run in super-wideband mode (plfreq = 32000). To set iSAC in channel-adaptive mode specify a rate of -1. This is the default behavior. In this mode, iSAC will use bandwidth information reported by the remote sender to choose the most appropriate bitrate for current network conditions.
- **G.729.1** has possible rates of: 8000, 12000, 14000, 16000, 18000, 20000, 22000, 24000, 26000, 28000, 30000, and 32000. The rate can also be set to an integer, 0 – 11, for a corresponding bit rate of 8000 – 32000.

8 Error Handling

This chapter describes error codes and gives recommendations on their handling.

Unless they are returning a useable value (many "Get" functions), all functions return 0 if the task was successfully performed and -1 otherwise. When a function returns -1, it simply means that the specific task could not be performed; it may not mean that a major error occurred.

For example, the function GIPSVEBase::GIPSVE_StartPlayout() returns -1 if the input value (channel) contains a negative number. However, it will also return -1 if the computer does not have a sound card installed.

To find out the specific reason to why a function has returned -1, GIPSVEBase::GIPSVE_LastError() must be called. It returns a positive integer corresponding to the last error that occurred in the GIPS VoiceEngine or -1 if no error has occurred. The positive integer values are referred to as error codes. Error codes are divided into the following three categories:

Error code	The severity of the error	Example error
100xx	VoIP call cannot be performed.	No sound card installed on the PC.
90xx	There is limited functionality.	The sound card does not support volume changes.
80xx	The task of the function is not performed.	VoiceEngine functions are called with invalid parameters.

For a description of specific error codes please see Appendix A: Error Codes.

Recommended Handling of Error Codes

The following table gives recommended actions based on the severity of the error.

Error code	Recommended action	Example error
100xx	Message-box displaying the specific error code and a request that the user should check the specific hardware/software not functioning. properly	No sound card installed on the PC.
90xx	Disable the feature not functioning properly.	The sound card does not support volume changes.
80xx	Nothing to be communicated to the end-user.	VoiceEngine functions are called with invalid parameters.

Appendix A: Error Codes lists the error codes and gives a possible reason that they occur. This can be used to create a message for the user that describes the problem and suggests a resolution.

Appendix A Error Codes

Appendix A describes all the possible error codes that can be returned when calling

```
int GIPSVEBase::GIPSVE_LastError();
```

Please see Recommended Handling of Error Codes for recommended handling of the error codes.

All error codes are declared in GIPSVEErrors.h.

NOTE: The base class for each sup-API is excluded in the table below to save space. As an example, instead of listing `GIPSVEBase::GIPSVE_StartListen`, only the API name (`GIPSVE_StartListen`) is given.

Error Code	Function Name/s	Problem Description	Possible Reason
8001	`GIPSVE_StartListen` `GIPSVE_StartSend`	No port has been set for listening/sending	These errors are usually caused one by of the following reasons: • Invalid input arguments have been sent to the function calls. • The parameters needed for performing the task have not been set. For example, a channel needs to be created before starting to play out on speaker. If that has not been done then error code 8013 will be returned.
8002	All functions with channel as input	Invalid channel number as function input	
8003	Version dependent	This functionality is not included in this version	
8004	`GIPSVE_GetCodec`	The input list number is larger/smaller than list size	
8005	Any call	The input variable is outside of the allowed range	

8006	`GIPSVE_SetLocalReceiver`
	`GIPSVE_SetSendDestination`
	`GIPSVE_StartListen`
	`GIPSVE_StartSend`
		The input port number is outside of the allowed range	

8007	`GIPSVE_SetSendCodec`	The input payload name is not correct	
8008	`GIPSVE_SetSendCodec`	The input frequency is not allowed	
8009	`GIPSVE_SetSendCodec`	The input payload type is outside of the allowed range or belongs to a codec not supported	
8010	`GIPSVE_SetSendCodec`	The input packet size is not supported	
8011	`GIPSVE_SetSendCodec`	Change of packet size during call is not supported	

Error Code	Function Name/s	Problem Description	Possible Reason
8012	GIPSVE_SetLocalReceiver GIPSVE_StartListen	VoiceEngine is already listening	
8013	GIPSVE_StartPlayout	The channel is not created yet	
8014	GIPSVE_StartPlayout	Maximum number of active channels is already reached.	
8015	GIPSVE_StartSend	Failed to prepare header for recording buffer	
8016	GIPSVE_StartSend	Failed to add buffer for recording	
8017	GIPSVE_StartPlayout	Failed to prepare header for playback buffer	
8018	GIPSVE_StartSend GIPSVE_SetSendDestination GIPSVE_SetSendTOS	VoiceEngine is already sending	
8019	GIPSVE_SetSendDestination GIPSVE_StartSend	The input IP address is invalid	
8020	GIPSVE_StartPlayout	VoiceEngine is already playing	
8021	GIPSVE_GetVersion	The input buffer is too small to hold all version info	
8022	GIPSVE_SendDTMF	The input DTMF tone number, length or level is invalid	
8023	GIPSVE_SetSendCodec	The input channels is not correct (codec parameter)	
8024	GIPSVE_SetRecPayloadType	Setting a new payload type for receiving failed	
8025	GIPSVE_SetEncryptionStatus	Encryption has not been initialized. That is an object to an encryption algorithm has not been passed to the VoiceEngine	
8026	GIPSVE_CreateChannel GIPSVE_SetNSStatus GIPSVE_GetNSStatus GIPSVE_SetAGCStatus GIPSVE_GetAGCStatus GIPSVE_SetECStatus GIPSVE_GetECStatus GIPSVE_SetConfStatus GIPSVE_GetConfStatus	VoiceEngine has not been initialized yet.	
8027	GIPSVE_SendDTMF	DTMF tones cannot be sent until the VoiceEngine has started sending media	

Error Code	Function Name/s	Problem Description	Possible Reason
8028	`GIPSVE_SetExternalTransport`	VoiceEngine is not compiled to support external transport protocol	
8029	`GIPSVE_SetLocalReceiver` `GIPSVE_SetSendDestination` `GIPSVE_StartListen`	VoiceEngine is compiled to support external transport protocol and these calls have no meaning.	
8030	`GIPSVE_StopSend`	Sound card failure	
8031	`GIPSVE_SetSendCodec`	Rate is not valid	
8032	`GIPSVE_ReceivedRTPPacket` `GIPSVE_ConvertRTPToFile`	RTP packet seems to be corrupt.	
8033	`GIPSVE_SetLocalReceiver`	No GQoS support.	
8034	`GIPSVE_ConvertRTPToFile`	Incoming timestamp parameter is out of sequence	
8035		See Appendix B.	
8036	`GIPSVE_SendDTMF`	Previous DTMF tone is still ongoing. Wait 100ms and try again	
8037	`GIPSVE_Init`	Incorrect expiry date.	
8038	`GIPSVE_StopListen`	Cannot stop listening when sending.	
8039	`GIPSVE_EnableIPv6`	Enabling IPv6 failed.	
8040	`GIPSVE_SetChannelOutputVolumePan`	No stereo support.	
8041-8060		Reserved for VoiceEngineATA. See VoiceEngineATA API Specification.	
8061-8080		Reserved for VoiceEngine for Symbian.	
8081	NA	Firewall traversal already enabled.	
8082		See Appendix B.	
8083	`GIPSVE_GetBuild` `GIPSVE_GetDevice` `GIPSVE_GetPlatform`	Not all info could be retrieved.	Too small buffer size.

Error Code	Function Name/s	Problem Description	Possible Reason
	GIPSVE_GetOS GIPSVE_GetLocalIP		
8084	GIPSVE_SetSendCodec	Send codec could not be set	
8085	GIPSVE_GetSendCodec	Error getting codec information	
8086	GIPSVE_GetNetworkStatistics GIPSVE_GetJitterStatistics GIPSVE_GetPreferredBufferSize GIPSVE_ResetJitterStatistics GIPSVE_GetRTPStatistics GIPSVE_GetRTCPStatistics GIPSVE_SetMinimumPlayoutDelay GIPSVE_SetChannelOutputVolumeScaling GIPSVE_GetChannelOutputVolumeScaling GIPSVE_SetVQMonStatus	NetEQ was not created successfully or caused an error	
8087		Reserved	
8088	GIPSVE_StartListen GIPSVE_GetVADStatus GIPSVE_ExternalRecordingInsertData GIPSVE_ExternalPlayoutGetData GIPSVE_StartRecordingPlayoutStereo GIPSVE_SetSoundDevices GIPSVE_SetSoundCardObject GIPSVE_NeedMorePlayData GIPSVE_RecordedDataIsAvailable GIPSVE_SetExternalTransport GIPSVE_ReceivedRTPPacket GIPSVE_ReceivedRTCPPacket GIPSVE_GetSourceInfo GIPSVE_EnableIPv6 GIPSVE_SetSourceFilter GIPSVE_GetSourceFilter GIPSVE_SetSendTOS GIPSVE_GetSendTOS GIPSVE_SetSendGQoS GIPSVE_GetSendGQoS GIPSVE_SendUDPPacket	VoiceEngine is in a state or mode in which the operation is invalid	External transport is enabled and the operation is invalid. Or the mode/interface in which the operation is valid has not been enabled.

Error Code	Function Name/s	Problem Description	Possible Reason
8089	GIPSVE_GetSystemCPULoad	Error getting CPU load	Application is not run in administrator mode.
8090	GIPSVE_GetPlayoutDeviceName GIPSVE_GetRecordingDeviceName GIPSVE_SetSoundDevices GIPSVE_ResetSoundDevice GIPSVE_SoundDeviceControl GIPSVE_SetSamplingRate GIPSVE_SetSoundCardObject GIPSVE_GetSoundCardBufferSize GIPSVE_SetWaveOutVolume GIPSVE_GetWaveOutVolume	Sound device caused an error	The sound device was not opened successfully or the operation caused an error
8091	GIPSVE_GetSpeechInputLevel GIPSVE_GetSpeechOutputLevel GIPSVE_GetSpeechInputLevelFullRange GIPSVE_GetSpeechOutputLevelFullRange	Error getting the speech level	
8092	GIPSVE_ExternalRecordingInsertData GIPSVE_SendUDPPacket	Error sending packet	
8093	GIPSVE_SetConferenceStatus	Error removing conference channel	
8094	GIPSVE_GetRecPayloadType	Error getting payload type for receive codec	
8095	GIPSVE_SetFECStatus	Error enabling FEC	
8096	GIPSVE_ExternalPlayoutGetData	Error getting play out data	
8097	GIPSVE_ResetCallReportStatistics GIPSVE_GetVADStatus GIPSVE_SetNSStatus GIPSVE_GetNSStatus GIPSVE_GetAGCStatus GIPSVE_GetECStatus GIPSVE_GetConfStatus GIPSVE_SetMetricsStatus GIPSVE_GetMetricsStatus GIPSVE_GetSpeechMetrics GIPSVE_GetNoiseMetrics GIPSVE_GetEchoMetics	GIPS VQE component caused an error or returned an invalid argument	
8098		See Appendix B	

Error Code	Function Name/s	Problem Description	Possible Reason
8099		See Appendix B	
8100	GIPSVE_PlayDTMFTone	VoiceEngine is not playing on any channel	
8101	GIPSVE_StartListen GIPSVE_SetSendGQoS	Unable to start receiving RTP packets	GIPSVE_SetLocalReceiver has not been called
8102	GIPSVE_GetLocalReceiver GIPSVE_SetSendDestination	Unable to retrieve information from the local socket layer	
8103	GIPSVE_SetSendDestination	Cannot define the sending socket structure	The provided mutli-cast address is invalid
8104	GIPSVE_StartSend GIPSVE_SetSendGQoS	Cannot start sending RTP packets	GIPSVE_SetSendDestination has not been called
8105	GIPSVE_SetExternalTransport	Unable to register the external transport object	Conflict with existing sockets on the receiving side
8106	GIPSVE_SetExternalTransport	Unable to register the external transport object	Conflict with existing sockets on the sending side
9001	GIPSVE_SetLocalReceiver	Cannot bind the socket for receiving RTCP packets	Network card hardware/software problem
9002	GIPSVE_SetMicVolume	Cannot set microphone level	Sound card problem
9003	GIPSVE_SetSpeakerVolume	Cannot set speaker volume	Sound card problem
9004	GIPSVE_Init	Cannot access sound card to change or retrieve recording level	Sound card problem
9005	GIPSVE_Init	Cannot access sound card to change or retrieve speaker volume	Sound card problem
9006	GIPSVE_GetSpeakerVolume	Cannot retrieve speaker volume	Sound card problem
9007	GIPSVE_StartListen	Cannot create thread for receiving RTCP packets	
9008	GIPSVE_Init	Cannot init AEC Cannot init AES	
9010	GIPSVE_StartSend	Cannot set TOS for outgoing stream. Everything else will work.	Forgot to set registry key?
9011	GIPSVE_GetVoIPMetrics GIPSVE_SetVQMonAlertCallback	Vqmon call fails enabled	Faulty parameters
9012	GIPSVE_SetVQMonStatus GIPSVE_GetVoIPMetrics	VQMon must be enabled first	

Error Code	Function Name/s	Problem Description	Possible Reason
9014	GIPSVE_EnableSRTPSend GIPSVE_DisableSRTPSend GIPSVE_EnableSRTPReceive GIPSVE_DisableSRTPReceive	Error in SRTP	
9016	GIPSVE_Release	Error releasing the interface reference	A reference to the interface has not been created
9017	GIPSVE_SetSendTOS GIPSVE_SetSendGQoS	Error setting TOS/GQoS	Trying to set TOS when GQoS is enabled or DSCP has already been set
9018	GIPSVE_SetConferenceStatus	Error adding the channel to the mix	The maximum number of conference channels has been reached
9019	GIPSVE_GetChannelMIMEParameters	Error adding data to buffer	The supplied buffer is too small
9020	GIPSVE_SetG729AnnexBStatus	Error setting Annex B mode	
9021	GIPSVE_GetG729AnnexBStatus	Error getting the Annex B mode	
9022		See Appendix B	
9023	GIPSVE_SetRTPKeepaliveStatus	Error setting the keep alive state	
9024	GIPSVE_SendDTMF	Error sending DTMF	
9025	GIPSVE_GetRemoteRTCP_CNAME	Error retrieving the remote CNAME	RTCP information has not been received
9026	NA	Error decrypting stream	The stream is not encrypted
9027	NA	Error encrypting stream	
9028	GIPSVE_GetRTCPStatistics	Error getting RTCP statistics	
9029	GIPSVE_SetSendDestination GIPSVE_SetSendGQoS	Unable to ensure that the socket layer supports GQoS	See Remarks section for the GIPSVE_SetSendGQoS API
9030	GIPSVE_SetLocalReceiver	Unable to bind socket to local address	Most likely the UDP ports are already in use
9031	GIPSVE_SetSendTOS	Failed to set TOS	Invalid TOS value
9032	GIPSVE_SetSendTOS	Failed to set TOS	Socket layer does not support modifying the TOS field in the IP header
10001	GIPSVE_StartPlayout	Undefined sound card error	Sound card problem
10002	GIPSVE_StartSend	Cannot open sound card for recording	Sound card problem

Error Code	Function Name/s	Problem Description	Possible Reason
10003	GIPSVE_SetLocalReceiver	Cannot bind the socket for receiving RTP packets	Network card hardware/software problem
10004	GIPSVE_StartPlayout	Write to sound card failed (invalid handle)	Sound card problem
10005	GIPSVE_StartPlayout	Write to sound card failed (no driver)	Sound card problem
10006	GIPSVE_StartPlayout	Write to sound card failed (no memory)	Sound card problem
10007	GIPSVE_StartPlayout	Write to sound card failed (header not prepared)	Sound card problem
10008	GIPSVE_StartPlayout	Write to sound card failed (still playing)	Sound card problem
10009	GIPSVE_StartPlayout	Write to sound card failed (undefined error)	Sound card problem
10010	GIPSVE_StartSend	Read from sound card failed (undefined error)	Sound card problem
10011	GIPSVE_StartListen	Cannot create thread for receiving RTP packets	Network card hardware/software problem
10012	GIPSVE_StartSend	Cannot start recording from sound card	Sound card problem
10013	GIPSVE_StartPlayout	Cannot open sound card for play back	Sound card problem
10014	GIPSVE_Init	Cannot start up windows sockets	Winsock 2 is not supported
10015	GIPSVE_StartSend	Cannot bind the socket for sending RTP packets	Network card hardware/software problem. The port number has not been specified with GIPSVE_SetLocalReceiver.
10016	GIPSVE_StartRecordingPlayout GIPSVE_StartRecordingMicrophone GIPSVE_ConvertWAVToPCM GIPSVE_StartPlayingFileLocally	This is not a valid file, or the file cannot be opened for reading.	The file does not exist, or is locked by another application.
10017	Any call	This is a time limited version of VoiceEngine and the time has expired.	
10018	Any call	You must first call GIPSVE_Authenticate with a valid password	
10019		See Appendix B	

Error Code	Function Name/s	Problem Description	Possible Reason
10020		See Appendix B	
10021	`GIPSVE_StartPlayingFileLocally`	Bad arguments	volume_scaling argument specified is either greater than 1.0 or less than 0.0. It can also happen if start_point is greater than stop_point or the file length. Another cause could be user entering same non zero values for both start_point and stop_point.
10022	`GIPSVE_SetSoundDevices`	Function is not supported on the platform	The platform does not run Linux
10023		See Appendix B	

10024	`GIPSVE_Init`
	`GIPSVE_CreateChannel`
	`GIPSVE_SendUDPPacket`
		Error allocating memory	Not enough memory to crate the object

10025	`GIPSVE_CreateChannel`
	`GIPSVE_GetVersion`
	`GIPSVE_ConvertPCMToWAV`
	`GIPSVE_ConvertWAVToPCM`
	`GIPSVE_ConvertPCMToCompressed`
	`GIPSVE_GetPlaybackPosition`
	`GIPSVE_SetG729AnnexBStatus`
	`GIPSVE_GetG729AnnexBStatus`
		Bad handle for used object	A bad handle was supplied to the function

10026	`GIPSVE_CreateChannel`	Error initializing the RTP/RTCP module	

Appendix B Runtime Error Codes

The following table described all possible runtime error codes. See Recommended Handling of Runtime Errors for more details.

Error code	Error name	Problem description	Possible reason
8035	VE_RECEIVE_PACKET_TIMEOUT	No packet received during the specified time (1-150 seconds).	
8082	VE_PACKET_RECEIPT_RESTARTED	Packet received again after packet timeout.	
8098	VE_RUNTIME_PLAY_WARNING	The playout audio can be distorted.	CPU overloaded
8099	VE_RUNTIME_REC_WARNING	The recorded audio can be distorted.	USB port overload or CPU overloaded
9022	VE_SATURATION_WARNING	The AGC has detected a saturation event despite minimum capture volume. Refer to `GIPSVE_SetAGCStatus()`.	Mic boost may be enabled.
10019	VE_RUNTIME_PLAY_ERROR	Playout fails	Playout device removed or CPU overloaded
10020	VE_RUNTIME_REC_ERROR	Recording fails	Recording device removed or CPU overloaded
10023	VE_REC_DEVICE_REMOVED	Directly precedes a VE_RUNTIME_REC_ERROR if it appears that a device has been removed. Only thrown on Windows.	