



## **PCBest Networks SIP SDK API Reference**

**(For SDK V2.10d)**

**Copyright 2011 PC Best Networks Inc.**  
[support@pcbest.net](mailto:support@pcbest.net)

## Index:

1	Introduction.....	7
1.1	key features and specifications .....	8
1.2	SDK Component Architecture .....	12
1.3	Samples in SDK.....	12
1.4	How to setup development environment.....	14
1.5	.NET users attention to use your application in 64-bit OS.....	15
1.6	Distribute your applications .....	17
2	SDK Programming Guide .....	18
2.1	Initialization code.....	18
2.2	Set up a timer to process GTAPI events .....	21
2.3	Working with GTAPI functions.....	21
2.4	Working with GTAPI events .....	22
2.5	Multiple channels supports.....	23
2.6	SDK Configurations.....	25
2.7	Sample code to implement basic incoming calls.....	26
2.8	SDK log.....	26
3	API Reference .....	27
3.1	Initialize SDK functions.....	27
3.1.1	StartServer .....	27
3.1.2	StopServer.....	27
3.2	Call Control Functions and Events.....	28
3.2.1	Send_Make .....	28
3.2.2	Send_MakeEx.....	29
3.2.3	Send_Answer.....	30
3.2.4	Send_HungUp deprecated (Use Send_HangUp) .....	30
3.2.5	Send_HangUp.....	31
3.2.6	Send_Hold .....	32
3.2.7	Send_Transfer.....	37
3.2.8	Send_TransferEx.....	38
3.2.9	Send_Redirect.....	39
3.2.10	Send_Accept.....	40
3.2.11	Send_Ring .....	41
3.2.12	Send_SessionProgress .....	41
3.2.13	Send_WaitForCall (new function from 2.05f) .....	42
3.2.14	On_RecvConnected .....	42
3.2.15	On_RecvOffered.....	43
3.2.16	On_RecvOfferedEx .....	44
3.2.17	On_RecvDialing .....	45
3.2.18	On_RecvRinging .....	45
3.2.19	On_RecvIdle (New Format) .....	46
3.2.20	On_RecvSessionProgress .....	47
3.2.21	On_RecvHolding .....	47
3.2.22	On_RecvTransferring.....	48
3.2.23	On_RecvNoAudio .....	49
3.2.24	GetChanCallID .....	49
3.2.25	GetChanCallFromTag.....	50
3.2.26	GetChanCallToTag .....	50
3.3	Channel Status Functions .....	51
3.3.1	Send_GetChanStatus.....	51
3.3.2	On_RecvStatus.....	52
3.3.3	Send_SetChanStatus .....	53

3.3.4	SetChanDir (New function since 2.05f).....	53
3.3.5	SetChanAudioDir.....	54
3.4	SIP Accounts Status Functions.....	55
3.4.1	Set SIP Accounts.....	55
3.4.2	SIP Accounts Register Event(On_RecvRegStatus).....	56
3.4.3	Send_GetRegStatus.....	56
3.4.4	SIPAccount_Register.....	57
3.5	Dynamical SIP Account Management (New Feature of SDK v1.71f).....	57
3.5.1	SIPAccount_Add.....	58
3.5.2	SIPAccount_Remove.....	59
3.5.3	SIPAccount_Count.....	59
3.5.4	SIPAccount_Index.....	59
3.5.5	SIPAccount_Handle.....	60
3.5.6	SIPAccount_Get.....	60
3.5.7	SIPAccount_Enable.....	61
3.6	DTMF Functions and Events.....	61
3.6.1	Send_PlayDTMFStr.....	62
3.6.2	Send_EnableDTMF.....	62
3.6.3	Send_DisableDTMF.....	63
3.6.4	On_RecvDTMFDone.....	63
3.6.5	On_RecvDTMFKeyDown.....	64
3.6.6	On_RecvDTMFKeyUp.....	65
3.6.7	SetChanDTMFType.....	65
3.6.8	GetChanDTMFType.....	66
3.7	CT_BUS Functions.....	66
3.7.1	Send_DuplexConnect.....	67
3.7.2	Send_DuplexDisconnect.....	67
3.7.3	Send_HalfConnect.....	68
3.7.4	Send_HalfDisconnect.....	68
3.7.5	Send_RTPDuplexConnect.....	69
3.7.6	Send_RTPDuplexDisconnect.....	69
3.7.7	Send_ProxyConnect and Send_ProxyDisconnect.....	70
3.8	Audio Control Functions and Events.....	70
3.8.1	Send_PlayAudio.....	71
3.8.2	Send_RecordAudio.....	71
3.8.3	Send_RecordAudio2.....	72
3.8.4	Send_AddAudio.....	73
3.8.5	Send_ClearAudio.....	74
3.8.6	Send_StopAudio.....	74
3.8.7	Send_StopAudioEx.....	75
3.8.8	Send_SetAudioFormat.....	75
3.8.9	Send_GetAudioStatus.....	77
3.8.10	Send_StartDXAudio.....	77
3.8.11	Send_StopDXAudio.....	78
3.8.12	Send_ResetDXAudio.....	79
3.8.13	Send_SetDXAudioVolume.....	79
3.8.14	On_RecvAudioPlayDone.....	80
3.8.15	On_RecvAudioRecordDone.....	80
3.8.16	On_RecvAudioStatus.....	81
3.8.17	On_RecvDXAudioStatus.....	82
3.9	MOH(Music On Hold) Functions and Events.....	83
3.9.1	Send_StartMusicOnHold.....	83
3.9.2	Send_StopMusicOnHold.....	84

3.9.3	Softphone “Hold” implementation .....	84
3.10	Conference Functions and Events .....	85
3.10.1	Send_StartConference .....	85
3.10.2	Send_StopConference.....	85
3.10.3	Send_SetChanInConference .....	86
3.10.4	CreateConf.....	87
3.10.5	DestroyConf.....	88
3.10.6	SetChanInConf .....	88
3.10.7	GetConfIndex .....	89
3.10.8	SetChanConfMask.....	89
3.11	Conference Audio Functions and Events .....	90
3.11.1	Send_ConfPlayAudio .....	90
3.11.2	Send_ConfRecordAudio.....	91
3.11.3	Send_ConfRecordAudio2.....	92
3.11.4	Send_ConfAddAudio .....	92
3.11.5	Send_ConfClearAudio.....	92
3.11.6	Send_ConfStopAudio .....	93
3.11.7	Send_ConfStopAudioEx.....	94
3.11.8	Send_GetConfAudioStatus .....	94
3.11.9	Send_ConfSetAudioFormat.....	95
3.11.10	On_RecvConfAudioStatus.....	96
3.11.11	On_RecvConfAudioPlayDone .....	97
3.11.12	On_RecvConfAudioRecordDone .....	98
3.12	Tone Detection Functions and Events .....	99
3.12.1	Send_AddTone .....	99
3.12.2	Send_ClearToneList .....	99
3.12.3	Send_StartToneDetection .....	100
3.12.4	Send_StopToneDetection.....	100
3.12.5	On_RecvToneDetected .....	101
3.13	VAD Functions and Events .....	101
3.13.1	On_VoiceActivityDetected .....	101
3.13.2	Send_EnableVAD .....	102
3.13.3	Send_DisableVAD .....	103
3.14	MWI(Message Waiting Indicator).....	103
3.14.1	On_RecvNotifySimpleMsgSummary .....	103
3.15	Instant Message Fonctions and Events.....	104
3.15.1	Send_MessageText .....	104
3.15.2	On_RecvMessageTextDelivered .....	105
3.15.3	On_RecvMessageText .....	105
3.16	Support for SIP/Presence .....	106
3.16.1	RecvSubscribeStatus.....	106
3.16.2	RecvNotifyPresence .....	107
3.17	RTP and DirecX Audio Data Events .....	107
3.17.1	On_RecvRTPPacket .....	108
3.17.2	On_SentRTPPacket.....	109
3.17.3	On_CaptureDXAudio .....	110
3.17.4	On_RenderDXAudio .....	111
3.17.5	On_RTPRawPacket .....	111
3.18	Softphone Assistant Functions .....	113
3.18.1	PlayNumTone .....	113
3.18.2	PlayLocalRingSound .....	113
3.18.3	PlayRemoteRingSound.....	114
3.18.4	PlayBusySound.....	115

3.18.5	StopSound.....	115
3.18.6	SetMicVolume .....	116
3.18.7	GetMicVolume.....	116
3.18.8	SetSpeakerVolume .....	117
3.18.9	GetSpeakerVolume .....	117
3.18.10	SetSpeakerMuteStatus .....	118
3.18.11	GetSpeakerMuteStatus.....	118
3.18.12	SetMicMuteStatus.....	119
3.18.13	GetMicMuteStatus .....	119
3.18.14	GetSoundDeviceCount deprecated .....	120
3.18.15	GetSoundDeviceName deprecated .....	120
3.18.16	GetRenderDeviceCount.....	121
3.18.17	GetRenderDeviceName .....	121
3.18.18	IsRenderDevicePrimary.....	122
3.18.19	GetCaptureDeviceCount.....	122
3.18.20	GetCaptureDeviceName .....	123
3.18.21	IsCaptureDevicePrimary.....	123
3.19	Network Functions .....	124
3.19.1	GetLocalIPCount .....	124
3.19.2	GetLocalIP.....	124
3.19.3	GetLocalNetworkIPAddress deprecated .....	125
3.19.4	GetSIPAddressInfo .....	125
3.19.5	GetDetectedNATType.....	126
3.19.6	GetMappedPublicSIIPAddress .....	127
3.19.7	GetMappedPublicSIIPPort .....	127
3.19.8	GetLocalSIPPort .....	128
3.19.9	GetLocalRTPPort.....	128
3.19.10	GetPeerSIIPAddress .....	128
3.19.11	GetPeerSIIPPort .....	129
3.19.12	GetPeerRTPIPAddress .....	130
3.19.13	GetPeerRTPIPPort .....	130
3.19.14	GetPeerSIPContactAddress .....	131
3.20	Error Event.....	131
3.20.1	On_RecvError.....	131
3.21	Timer functions .....	133
3.21.1	StartTimer .....	133
3.21.2	StopTimer .....	133
3.21.3	OnTimer.....	134
3.22	SDK License Functions.....	134
3.22.1	SetAppName.....	135
3.22.2	GetLicTo.....	135
3.22.3	Tags for license .....	136
3.23	Human voice or Answering Machine Detection .....	137
3.23.1	On_DetectHumanVoiceDone.....	137
3.24	Other functions.....	137
3.24.1	GetChanAudioCodec .....	137
3.24.2	GetChanLastMsgCode.....	138
3.24.3	GetChanLastMsgText .....	139
3.24.4	SetChanUserData.....	140
3.24.5	GetChanUserData .....	141
3.24.6	SetMainWnd.....	141
3.24.7	GetTotalChannelNumber .....	142
3.24.8	GetChanAudioRecordFileName .....	142

	3.24.9	SetChanCallExtraSIPHeader .....	143
	3.24.10	SetChanAudioLevel.....	144
	3.24.11	On_SIPMsg .....	144
4	Proxy API.....		145
	4.1	Initialize Proxy Site.....	145
	4.1.1	InitProxySites.....	145
	4.2	Free Proxy Site.....	146
	4.2.1	FreeProxySites .....	146
	4.3	Add, Delete, Change or Disable Proxy Site User.....	147
	4.3.1	ProxySetUserInfo.....	147
	4.3.2	ProxyDisableAllUsers .....	148
	4.3.3	ProxySetUserMsg .....	149
	4.3.4	ProxyUpdateUserState .....	150
	4.4	Events.....	150
	4.4.1	On_ProxyUserRegistered .....	150
	4.4.2	On_ProxyNewCallSession.....	152
	4.4.3	On_ProxyCallSessionStatus.....	153
	4.4.4	On_ProxyCallTransaction.....	155
	4.4.5	ProxySetNewCallSessionAddr .....	156
	4.4.6	On_ProxyUserSubscribed.....	157
5	SDK Configuration Tags.....		158

# 1 Introduction

PCBest Networks provides NO.1 Windows VOIP development kits to business customers. Its SIP SDK is the most powerful, stable and easy to use VOIP tool in Windows platform, and it is suitable for both SIP softphone and server applications. The SDK provides rich samples for all kinds of programmers, including .NET, C++, Borland C++, Delphi, VB6 and others. It is also stress tested and has no memory leaking, which makes it extremely good for server applications, especially Windows XP Embedded.

The softphone SDK package also has excellent sound quality, and supports G729 codec. An fantastical feature is it supports switching sound devices in a live call. This makes call-center agents switch from speaks to USB headset phone so easy without cutting off a call. Another good feature is it supports call recording, which enables service quality tracking available for agents.

Majorly the SDK provides four interfaces to programmers.

1. C++ headers and lib.
2. Standard DLL interface. (GTAPI\_Face2.h is for all interfaces functions)
3. .NET assembly for .NET programmers. (a .net dll to wrap standard dll interfaces, but it is made by c# with source code.)
4. An ActiveX(ocx) without any GUI.

All four above interfaces have similar methods and functions, so you can switch among them very easily, or you can choose one that mostly suits your development environment.

We have another great product, "[Robust SIP ActiveX phone](#)", which is a good software for customers who want to integrate softphones in their applications or web sites with less development time. This component provides even simple interfaces but it is still very flexible to change all kinds of settings. It has a phone GUI, but it can be hidden into background in your application or web pages and work with your own phone interfaces.

Try our software before you make any purchases. It is free to download, and free to try before you purchase.

We have customers, who tried or bought others, but finally stay with us because:

1. We provide excellent customer supports. Our experts will answer your all kinds of technical questions in first time, also provide solutions.
2. We are dedicated to our customers. We add new features that customers need.

3. Our software is more stable, and better for long run. It doesn't only work for a few of times. It always works, and has been tested with hundreds of hardware devices, SIP ITSP providers, and software.
4. Well designed and organized. Pure OOP and events driver designed. The SDK makes developments so easy, so customers can focus on their business logic, not always worrying about call status and struggling in SIP messages.
5. Rich features and samples, and can do all kinds of complex operations by just a simple command, function, or setting.

Our contact information for support:

**Email:** [support@pcbest.net](mailto:support@pcbest.net)

**Toll Free(USA & Canada):** 1-866-807-0528

**Local and International:** 1-613-239-4278

### 1.1 key features and specifications

SIP stack	PCBest Network SIP Stack(completely own and developed by PCBest. We can decode any additional features in SIP message for your SIP Project, or fix any problems that may exist in the SIP core.)	Very stable and compact size
Compatible SIP Servers, Proxy and PBX	Full compatible with Open SER, Asterisk, Cisco CallManager, Audio Codes, 3CX, Radvision, Rainbow and more others SIP platforms.	
Compatible SIP Hardwares	Full compatible with DLink, Audio codes, Grandstream, Cisco, Huawei, other major SIP hardware phones and PBXs.	
Supported Platforms	MS Windows(98/2000/XP/2003/Vista/2008) Softphone SDK needs DirectX 8+	
<b>Programming interfaces</b>	C++ head files and lib .NET assembly(managed interfaces) ActiveX control Standard DLL Interface	



## PC Best Networks SIP SDK API Reference

<b>Supported development tools</b>	MS Visual Studio 2003/2005/2008(C#, VB.NET, J#, ASP.NET) MS Visual Studio 6(VC6, VB6, ...) Borland C++ 5/6/7 Delphi 6/7 CodeGear Delphi 2007 CodeGear C++ Builder 2007 Java, JavaScript, HTML, and other windows development tools which support ActiveX control	
Audio call	Yes	
Audio codecs	G.711 uLaw/aLaw, G726, GSM, iLBC, Speex. G729(optional).	
Call transfer	YES	
Call hold	YES	
<b>Audio Record (Dynamically turn on in a live call)</b>	YES (Record Audio Mix )	Record the audio data and save as WAV files
Wav file play and record	YES	
Message Waiting Indicator (MWI)	YES	Implemented as RFC 3842
Supported SIP Methods	REGISTER, INVITE, CANCEL, INFO, BYE, ACK, REFER, SUBSCRIBE, OPTIONS, NOTIFY, MESSAGE, UPDATE	
RFC supported	RFC 3261, RFC 3665, RFC 2833, RFC 2327, RFC 3264, RFC 3550, RFC 3263, RFC 3891, RFC 3515, RFC 3420, RFC 3892, RFC 3265, RFC 3666, RFC 3489, RFC 3920, RFC 3921, RFC 3922, RFC 3923, RFC 4622, RFC 4854, RFC 4979, RFC 3842,	

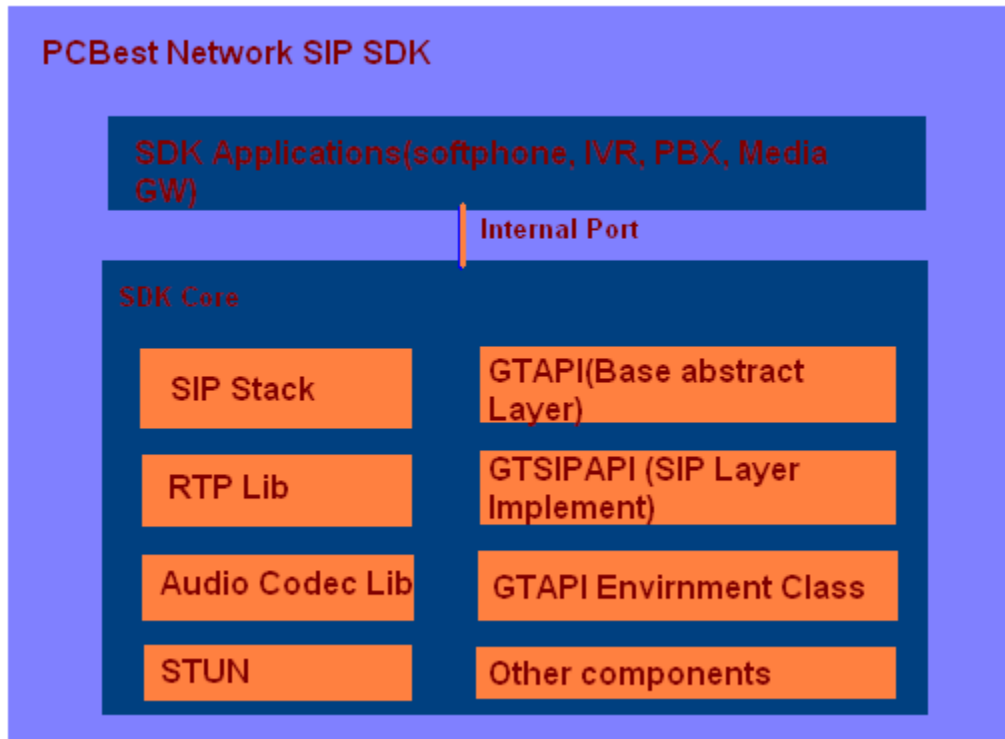
## PC Best Networks SIP SDK API Reference

Authentication	HTTP Basic Digest Authentication	
DTMF supported	RFC2833 / SIP INFO / Inband / Auto	
Multiple Calls(Simultaneous channels)	Yes	
Basic Telephony	DND(Do Not Disturb, Auto answer, Redial, 302 Redirect Call)	
<b>RTP Package Access</b>	Support access incoming and outgoing RTP audio stream directly. And support <b>change RTP audio stream to integrate TTS and ASR engine</b>	*****Very powerful feature.
<b>DirectX Audio Stream Access</b>	Yes. Can Access and change the DirectX audio on the middle way on both play and record direction	*****Very powerful feature.
<b>Support dynamically change sound devices during a live call</b>	YES	*****Very powerful feature. Good for call center agent softphone to switch between Speaker and USB headset without cutting off a call
Microphone & Speaker Device Selector	YES	
Microphone & Speaker Volume control	YES	
SIP UDP Support	YES	
Acoustic Echo Cancellation	YES	
<b>Conference</b>	YES	*****

## PC Best Networks SIP SDK API Reference

<b>Inbound Voice Activity Detection(VAD)</b>	YES	***** Can be used to detect human or answer machine voice
Outbound Proxy supported	YES	
STUN supported	YES	
Jitter Buffer	YES	
Free product version upgrades	YES	We provide 3 months free upgrade
Private Encrypt	YES	
Channel Timer	YES	
GUI customization	YES	
<b>Tone Detection</b>	YES	Good for fax detection and ring tone detection

## 1.2 SDK Component Architecture



## 1.3 Samples in SDK

There are rich samples in SDK\samples folder. The following list is for current version of SDK 1.71.

### Samples for server applications:

#### SDK\Samples\Server\vc\AudioRecord

A C++ sample for accessing RTP package to do audio recording. Note: *This is only a demo for accessing RTP stream. It doesn't mean this is the only way to record audio. We actually do NOT recommend you to use this method to record audio, because SDK does provide a function(RecordAudio) to achieve this purpose.*

#### SDK\Samples\Server\vc\SIPOutboundBroadcast

An outbound broadcasting SIP dialer made by C++.

**SDK\Samples\Server\vc\SIPServerApp**

**A very important demo for server application developers.** C++ code. It demos Play Audio, Record Audio, Echo Test, DTMF Detection, FullDuplex and HalfDuplex connect, VAD, Conference, and other server features. VB6 and VB.NET code of SIPServerApp are also available in SDK samples.

**SDK\Samples\Server\vb.net\VBSIPServerApp**

VB.NET code for SIPServerApp. It demos Play Audio, Record Audio, Echo Test, DTMF Detection, FullDuplex and HalfDuplex connect, VAD, Conference, and other server features.

**SDK\Samples\Server\c#\CSharpSIPServerApp**

C# code for SIPServerApp. It demos Play Audio, Record Audio, Echo Test, DTMF Detection, FullDuplex and HalfDuplex connect, VAD, Conference, and other server features.

**SDK\Samples\Server\c#\CSHumanDetect**

C# sample code for detecting human voice or answering machine

**SDK\Samples\Server\c#\BulkDialer**

C# sample code for outbound dialer.

**SDK\Samples\Server\vc\StressTest**

A C++ code used to do internal stress test for SDK.

**SDK\Samples\Server\OCX\VB6**

VB6 version of SIPServerApp

**SDK\Samples\Server\OCX\delphi\SIPStressServer and SIPStressClient**

Delphi codes written by one of our Delphi customers to do stress test

**Samples for softphone applications:**

**SDK\Samples\Softphone\GTF2Test**

A C++ sample to show how to access SDK by using standard DLL interfaces. It implements a simple SIP phone.

**SDK\Samples\Softphone\GTSimplePhone**

A C++ sample to use C++ header and lib. It also implements a simple SIP phone.

**SDK\Samples\Softphone\OCX\BCB6**

Borland C++ version of simple SIP phone.

**SDK\Samples\Softphone\OCX\Delphi6**

Delphi version of simple SIP phone

**SDK\Samples\Softphone\OCX\VB6**

VB6 version of simple SIP phone

**SDK\Samples\Softphone\c#\CSharpSIPPhone**

**C# version of a complete SIP phone.** This is a sample that covers most softphone features. Every developer who wants to use our SIP SDK should somehow review this code, and learn how to make a complete softphone.

**SDK\Samples\Softphone\vb.net\VBSIPPhone**

**VB.NET version of a complete SIP phone.** This is a sample that covers most softphone features. Every developer who wants to use our SIP SDK should somehow review this code, and learn how to make a complete softphone.

**Samples for PBX or Proxy server applications:**

**SDK\Samples\ProxyPABX\vb.net\VB-PABX**

A simple vb.net PBX sample

**SDK\Samples\ProxyPABX\vb.net\VBProxyCalls**

A simple vb.net sample for proxy calls.

**SDK\Samples\ProxyPABX\vb.net\VBSIPRegisterServer**

A simple vb.net sample for SIP registration server

**SDK\Samples\ProxyPABX\c#\CS-PABX**

A simple C# PBX sample

**SDK\Samples\ProxyPABX\c#\ProxyCalls**

A simple C# sample for proxy calls.

**SDK\Samples\ProxyPABX\c#\SIPRegisterServer**

A simple C# sample for SIP registration server

**SDK\Samples\ProxyPABX\OCX\VB6-PBX**

A vb6 sample for PBX by using ActiveX interface

There are other samples may not be listed here. Please refer to the sample source code.

## 1.4 How to setup development environment

### For C++ developers

1. Please set SDK\inc folder as C++ include directory.
2. Set SDK\lib folder as C++ link lib folder

3. Set project link with GTAPI.lib
4. Set project running in SDK\Bin folder.(or Set .exe file is generated into SDK\bin)
5. Derive a class MyGTAPIEnv from CGTAPIEnv class, and include "gtapi\_face.h" in this file.
6. Trying to overwrite some virtual functions for events. For example, On\_RecvConnected, On\_RecvDialing, ..... Those functions are defined in GTNetCmdClient.h.
7. Read other C++ samples and finish your code.

If you use VC 8.0 and VC9.0(Visual Studio 2005 and 2008), you need define **\_USE\_32BIT\_TIME\_T** flag to get it to link to gtapi.lib.

#### **For .NET developers**

1. Add reference to SDK\bin\GTAPIASM.dll
2. Set project to run in SDK\bin folder
3. Derive a class MyGTAPIEnv from GTAPIASM.GTAPIEnv
4. Trying to overwrite some events function like On\_XXXX to implement your call logics.
5. Read other .net samples code and finish your coding.

#### **For ActiveX(OCX) developers**

1. Use regsvr32 to register the ocx in bin folder. Like this: regsvr32 GTSIPCtrl.ocx
2. In your development tool, add the reference to the ActiveX control.
3. Add an ActiveX object into your form
4. Trying to overwrite events of ActiveX to implement your logic.
5. Refer to other OCX codes and finish your code.

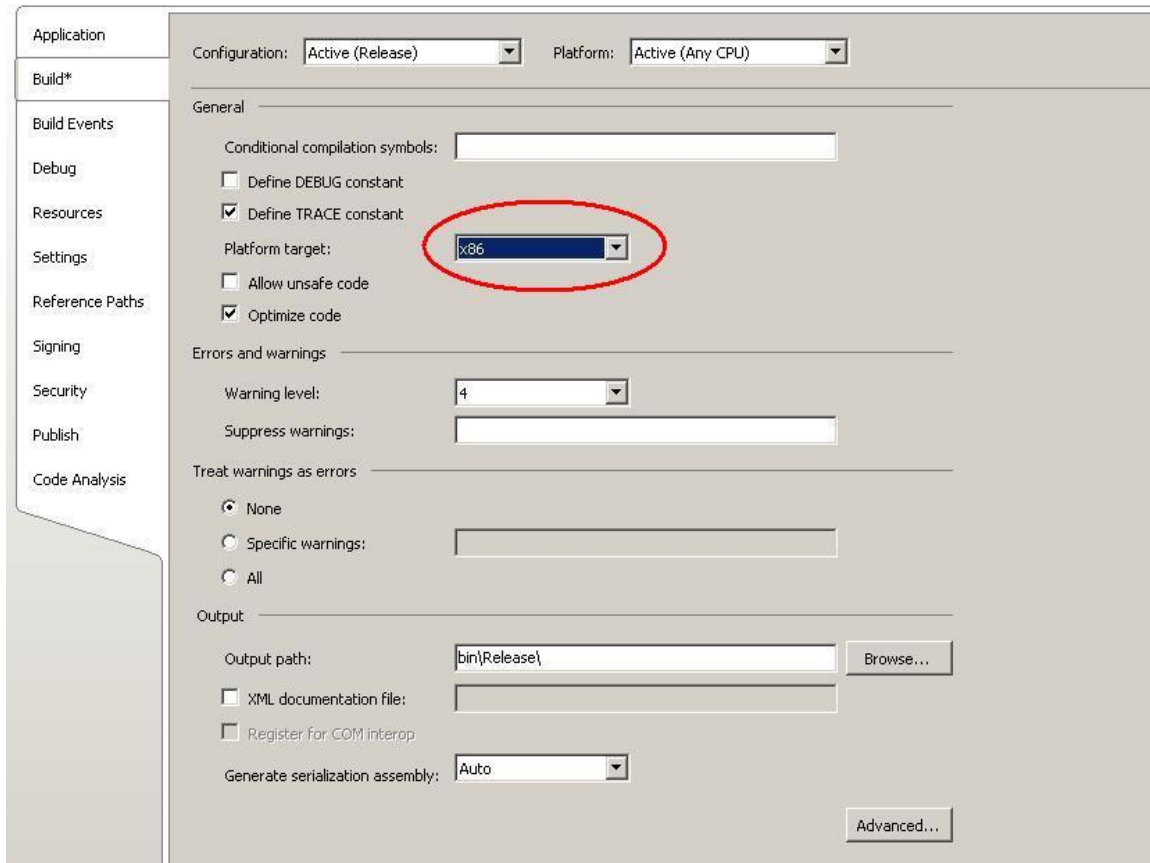
#### **For Standard DLL Interface Developers**

1. Open GTAPI\_FACE2.h for all definitions of functions.
2. Port the interfaces into your language, for example, Java or PHP.
3. Finish your code.

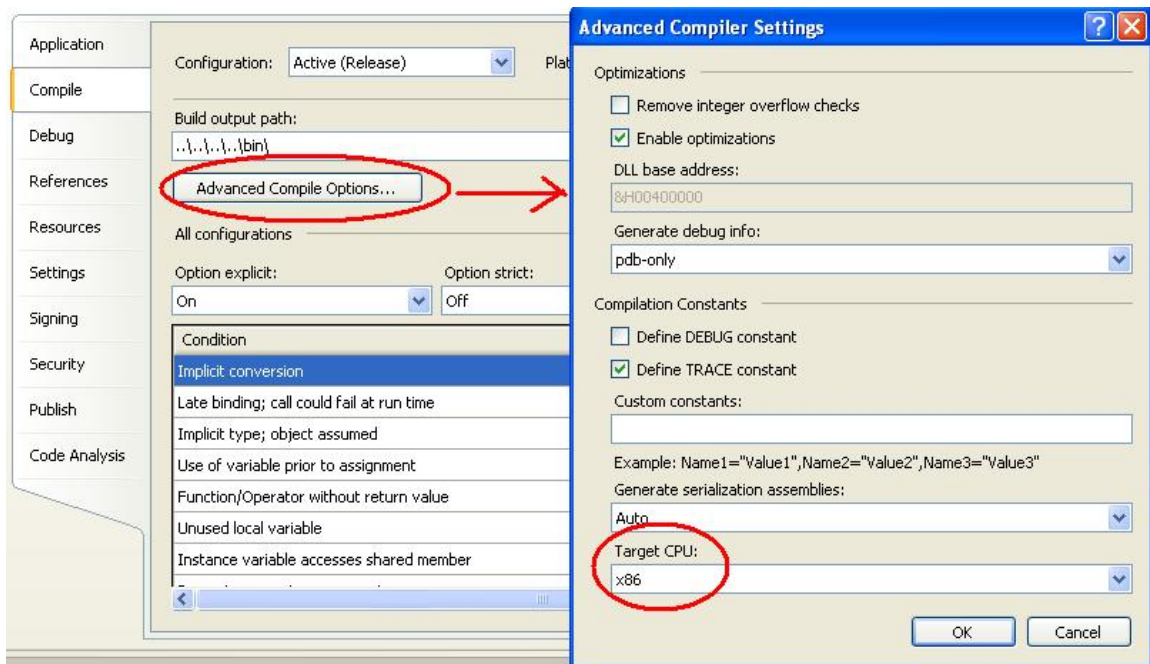
## **1.5 .NET users attention to use your application in 64-bit OS**

PCBest SIP SDK is 32bit API, but it can run on 64-bit OS without any problem. If you want to run your .NET application(C#, VB.NET, and C++) in 64-bit OS, you may need to set your project to target x86 platform, so 64-bit OS will know your application is for 32bit OS only and run WOW for your application.

**C# Project** should be set like the picture showing below:

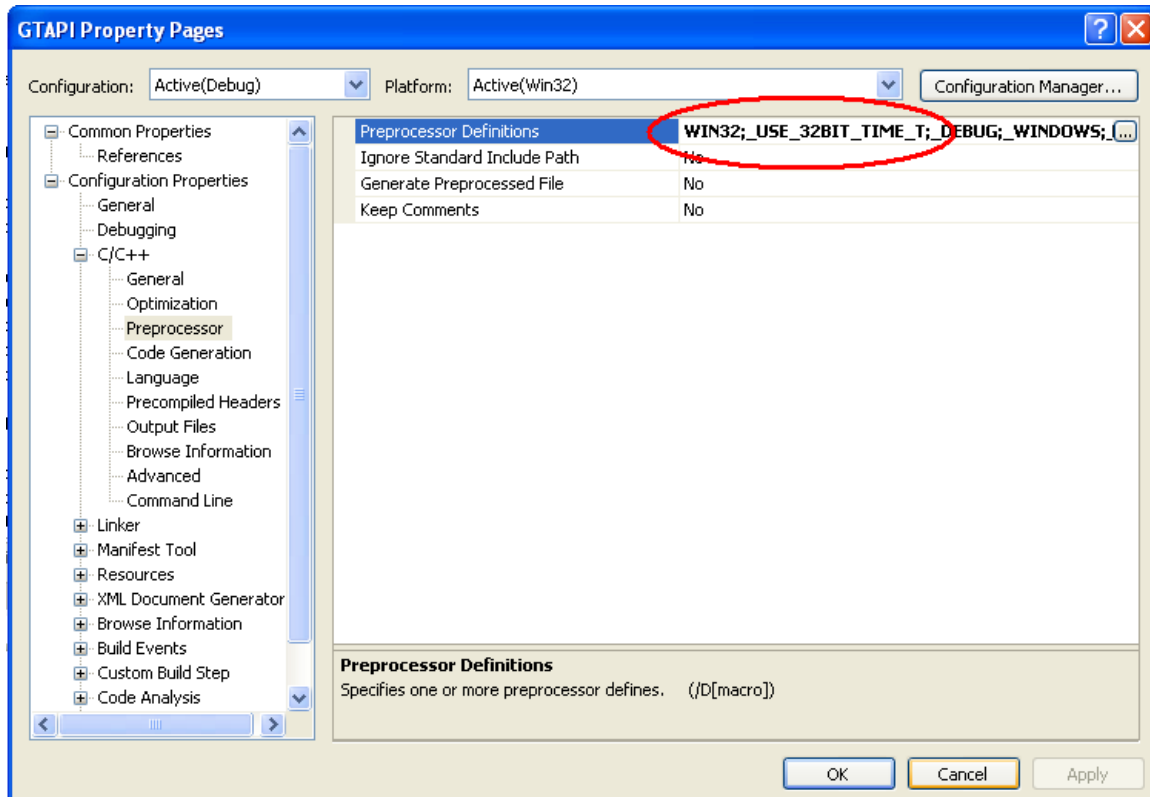


VB.NET project should have settings like this:





**C++ projects** should define WIN32 and \_USE\_32BIT\_TIME\_T:



## 1.6 Distribute your applications

### C++ Applications:

You should put all dlls in SDK\bin with your exe file when you distribute your application into destination machine.

### .NET Applications:

The same as C++ application, you should copy all dlls in SDK\bin with you exe file into same folder of destination machine.

### OCX(ActiveX) Applications:

The OCX file, gtsipctrl.ocx, is the only file you need to copy to the destination machine with your application, and of course, it is required to register in Windows by regsvr32. Command line to register it: regsvr32 gtsipctrl.ocx

## 2 SDK Programming Guide

### 2.1 Initialization code

No matter what language you are using, we suggest you to use the following SDK initialization code for your programming convenience.

#### 1. C++ developers

```
MyGTAPIEnv *g_pEnv = 0;
void InitSIPServer()
{
    if(g_pEnv) //already Initialized
        return;

    g_pEnv = new CGTSIPServerAppEnv;
    g_pEnv->SetMainWnd(m_hWnd);
    g_pEnv->CFG_SetValue("gtsrv.sip.xxxx.xxxx", "xxx");
    g_pEnv->CFG_SetValue(.....,.....);
    .....
    g_pEnv->StartServer();
}

void FreeSIPServer()
{
    if(!g_pEnv) //already free
        return;

    g_pEnv->StopServer();
    delete g_pEnv;
    g_pEnv = 0;
}
```

#### 2. .NET developers

VB.NET Code:

```
Public Sub InitSIPServer()
    env = New MyGTAPIEnv
    env.CreateEnv()
```

```

env.SetMainWnd(Me.Handle.ToInt32())

    //your all configuration code
    env.CFG_SetValue("gtsrv.sip.xxx.xxx.xxx", "xxx")
..... //other settings

If env.StartServer() = False Then
    //not successfully initied
Else
    //initied
End If
End Sub

Public Sub FreeSIPServer()
    env.StopServer()
    env.DestroyEnv()
    env = Nothing
End Sub

```

C# code:

```

public void InitSIPServer()
{
    if (env != null) return;

    env = new MyGTAPIEnv();
    env.CreateEnv();
    env.SetMainWnd(Handle.ToInt32());

    env.CFG_SetValue("gtsrv.sip.server.model", "0"); //client phone
    ..... //other settings

    if (!env.StartServer())
    {
        //not successfully started
    }
    else
    {
        //successfully initied
    }
}

public void FreeSIPServer()
{
    if(env != null)
    {
        env.StopServer();
    }
}

```

```

        env.DestroyEnv();
        env = null;
    }
}

```

### 3. OCX developers

VB6 code:

```

Public Sub InitSIPServer()
    GTSIPAPI1.CreateEnv
    GTSIPAPI1.CFGSetValue "gtsrv.sip.xxx.xxx", "xxx"
    GTSIPAPI1.StartServer
End Sub

```

```

Public Sub FreeSIPServer()
    GTSIPAPI1.StopServer
    GTSIPAPI1.DestroyEnv
End Sub

```

### 4. DLL standard interface developers

C++ Code:

```

void InitSIPServer()
{
    GTAPI_CreateEnv();
    GTAPI_SetMainWnd(m_hWnd);

    GTAPI_CFG_SetValue("gtsrv.sip.server.model", "0"); //client phone
    ..... //other settings

    if (!GTAPI_StartServer())
    {
        //not successfully started
    }
    else
    {
        //successfully initied
    }
}

void FreeSIPServer()
{
    GTAPI_StopServer();
    GTAPI_DestroyEnv();
}

```

```
}
```

## 2.2 Set up a timer to process GTAPI events

**Only** OCX and .NET developers need to set up a timer to process gtapi events. Why? SDK has many low level threads to generate events. For OCX and .NET events mechanize, the event functions or virtual functions of Env class have to be called by Windows GUI application's GUI thread. The OCX and .NET will queue the events when events arrived, and the Timer thread(in On\_timer event) will dequeue the events out, and fire the events or execute the virtual functions. This avoids the issue with threading problem or potential exe running crash.

### .NET developers

Add a timer into form, and set the Interval to 200ms-500ms. Add an event for timer tick:

```
On_Timer()
{
    env.ProcessGTAPIEvent ();
}
```

### OCX developers

Add a timer into form or dialog, and set the Interval to 200ms-500ms. Add an event for timer tick:

```
On_Timer()
{
    GTSIPAPI1.ProcessGTAPIEvent()
}
```

## 2.3 Working with GTAPI functions

The SIP SDK provides a lot of functions for application level. Some of them have an channel index as the first parameter. Those functions are designed for operate the specific channel.

### Channel index

Start from 0 to MaxChannelNumber -1

You can use env.GetChannelCount() to retrieve the max number of channels. Then how to specify the number of channels to use in the application?

There are 3 configuration tags for them

gtsrv.sip.channum.per.span (can be **1-30**)

gtsrv.sip.spannum.per.board (can be **1-16**)

gtsrv.sip.boardnum.per.server (can be **1-20**)

The total of channels is the multiple result of those 3 tags.

A sample which uses channel index is Send\_Make(int ch, ...)

Send\_Make can make an outbound call on the channel ch.

### Asynchronized functions

In above sample, Send\_Make will return void. You may ask why most of gtapi functions are returning void. Because most of functions are asynchronized functions, it means you won't know result until you get events for it. In order to make the programming simple, we designed those functions as asynchronized, so developers don't need to deal with multi-threading programming. It is also efficient to deal with multiple channels(or huge amount of channels) in server applications. Developers can design their business call status machine and process all the channels in the events(one thread). Here we will list the sample process of Send\_Make

Possible Send\_Make results:

1. Channel is not available to call out, so the event you will receive after Send\_Make is On\_RecvError
2. Call dialed, but it cannot make through. You will receive On\_RecvDialing, then On\_RecvIdle. After that, you can retrieve channel last message code for why it didn't go through.
3. Call is connected. The events you will receive is On\_RecvDialing, On\_RecvConnected

## 2.4 Working with GTAPI events

The SIP SDK use event mechanism to notify above applications about the status, events, and errors. Different language has different implements of events.

**C++:** Virtual functions with prefix On\_ in CGTAPIEnv class. If you want to write your own events handle, you have to derive a class from CGTAPIEnv, and overwrite the virtual functions.

**.NET:** Virtual functions with prefix On\_ in GTAPIASM.GTAPIEnv class. If you want to write your own events handle, you have to derive a class to overload the virtual functions.

**OCX:** Events interfaces. You have to implement the events in your own language development environment.

**DLL:** Provides functions to set callback functions.

## 2.5 Multiple channels supports

Every function or event related to channel will have a parameter CH(Channel ID). Channel ID is from 0 to maxChannelNumber – 1. The steps to setup multiple channels application are:

### 1. Set the number of channels you want app to have:

```
env.CFG_SetValue("gtsrv.sip.boardnum.per.server", "1");
env.CFG_SetValue("gtsrv.sip.spannum.per.board", "1");
env.CFG_SetValue("gtsrv.sip.channum.per.span", "4"); //here we use 4
channels.
```

As this setting will change threading model of SDK application, so please be careful with it. We don't recommend you set "gtsrv.sip.channum.per.span" more than 16.

#### Suggested Configuration of CHANNELS, SPANS, and Boards:

Channels	gtsrv.sip.channum .per.span	gtsrv.sip.spannum .per.board	gtsrv.sip.boardnum.p er.server
4	4	1	1
8	8	1	1
16	8	2	1
24	8	3	1
32	8	4	1
40	8	5	1
64	8	8	1
128	8	8	2
256	8	8	4
512	16	8	4

### 2. Deal with channel id in functions and events in your code

You may need to setup a simple status machine to process multiple channels.

### 3. Functions relate to channel

//Call control command

```
Send_Make
Send_Answer
Send_HungUp
Send_Hold
Send_Transfer
Send_Redirect
Send_Accept
Send_Ring
```

//channel status functions. see the definition of On\_RecvStatus below.

Send\_SetChanStatus  
Send\_GetChanStatus

//DTMF Handle

Send\_PlayDTMFStr  
Send\_EnableDTMF  
Send\_DisableDTMF

//CT\_BUS Handle

Send\_DuplexConnect  
Send\_DuplexDisconnect  
Send\_HalfConnect  
Send\_HalfDisconnect

//audio control command

Send\_PlayAudio  
Send\_RecordAudio  
Send\_AddAudio  
Send\_ClearAudio  
Send\_StopAudio  
Send\_SetAudioFormat  
Send\_GetAudioStatus

Send\_StartDXAudio  
Send\_StopDXAudio  
Send\_ResetDXAudio

//music on hold

Send\_StartMusicOnHold  
Send\_StopMusicOnHold

//conference

Send\_StartConference  
Send\_StopConference  
Send\_SetChanInConference

#### **4. Events relate to channel**

On\_RecvConnected  
On\_RecvOffered  
On\_RecvDialing  
On\_RecvRinging  
On\_RecvIdle  
On\_RecvHolding



On\_RecvTransferring  
 On\_RecvAudioPlayDone  
 On\_RecvAudioRecordDone

On\_RecvAudioStatus  
 On\_RecvRTPPacket  
 On\_SentRTPPacket  
 On\_CaptureDXAudio  
 On\_RenderDXAudio

On\_VoiceActivityDetected  
 On\_RecvDTMFDone  
 On\_RecvDTMFKeyDown  
 On\_RecvDTMFKeyUp  
 On\_RecvRegStatus

On\_RecvError

## 2.6 SDK Configurations

The SDK uses one unique interface to set or retrieve configuration items. Some items have to be set before the SDK application starts. Some of them are dynamic, and can be set later when the app is running.

Configuration Functions:

### **C++ and .NET:**

CFG\_GetValue //get string value of a configuration item  
 CFG\_GetIntValue //get integer value of a configuration item  
 CFG\_GetLongValue //get long integer value of a configuration item  
 CFG\_SetValue //set string value for a configuration item

### **OCX:**

CFGGetValue //get string value of a configuration item  
 CFGGetIntValue //get integer value of a configuration item  
 CFGGetLongValue //get long integer value of a configuration item  
 CFGSetValue //set string value for a configuration item

### **DLL:**

GTAPI\_CFG\_GetValue //get string value of a configuration item  
 GTAPI\_CFG\_GetIntValue //get integer value of a configuration item  
 GTAPI\_CFG\_GetLongValue //get long integer value of a configuration

item

GTAPI\_CFG\_SetValue //set string value for a configuration item

For the full list of configuration items, please refer to the configuration tag list.

## 2.7 Sample code to implement basic incoming calls

Please look at the samples in SDK folder.

## 2.8 SDK log

SDK log can be enabled by the following code:

```
CFG_SetValue("gtsrv.log.level", "4")
CFG_SetValue("gtsrv.log.filename", "c:\\vbsimplephone.txt")
```

Log level definition:

GT_LOG_DISABLE	0
GT_LOG_ERROR	1
GT_LOG_ALERT	2
GT_LOG_DEBUG	3
GT_LOG_INFO	4

The bigger number of log level, the more info you will get in log.

## 3 API Reference

### 3.1 Initialize SDK functions

#### 3.1.1 StartServer

**Description:** Init SIP SDK and Start. After this function is used, all necessary steps have been taken, and SIP port should be opened to accept incoming calls. This function may take a while to finish, because it needs to check your network structure, and startup channels.

**Format:**

C++: bool StartServer()  
.NET: bool StartServer()  
OCX: bool StartServer()  
DLL: bool GTAPI\_StartServer()

**Parameters:**

**Return:**

True if succeed, Otherwise false.

**Sample code:**

```
//Start SIP SDK Server  
Env.StartServer();
```

#### 3.1.2 StopServer

**Description:** Stop SIP SDK Server.

**Format:**

C++: bool StopServer()  
.NET: bool StopServer()  
OCX: bool StopServer()  
DLL: bool GTAPI\_StopServer()

**Parameters:****Return:**

True if succeed, Otherwise false.

**Sample code:**

```
//Stop SIP SDK Server
Env.StopServer();
```

## 3.2 Call Control Functions and Events

### 3.2.1 Send\_Make

**Description:** Make a call out. The SDK will issue On\_RecvDialing event to notify that the channel is dialing.

**Format:**

```
C++: void Send_Make(int ch, const char* calledNum, const char* callerNum)
.NET: void Send_Make(int ch, string calledNum, string callerNum)
OCX: void SendMake(int ch, BSTR calledNum, BSTR callerNum)
DLL: void GTAPI_Send_Make(int ch, const char* calledNum, const char*
callerNum)
```

**Parameters:**

Ch: Channel Index

CalledNum: Called number address in SIP format. Must be string like:  
“<sip:1234@abc.com>”, “<sip:5678@10.98.1.1:5060>”.

CallerNum: Caller id in SIP format. It can be “”.

**Return:**

null

**Sample code:**

```
//Make a call out on the first channel(0) to user 1234 on pcbest.net.
Send_Make(0, “<sip:8888@pcbest.net>”, “”);
//Make a call out on the first channel(0) to user 1234 on the machine 10.98.1.1:5070.
Send_Make(0, “<sip:1234@10.98.1.1:5070>”, “”);
```

### 3.2.2 Send\_MakeEx

**Description:** Make a call out. The SDK will issue On\_RecvDialing event to notify that the channel is dialing.

**Format:**

C++: void Send\_MakeEx(int ch, const char\* sCallee, const char\* sCaller, const char\* sURI, const char\* sContact, const char\* sAuthName, const char\* sAuthPassword, const char\* sDestIP, unsigned short nDestPort);

.NET: void Send\_Make(int ch, string calledNum, string callerNum, string sURI, string sContact, string sAuthName, string sAuthPassword, string sDestIP, ushort nDestPort)

OCX: void SendMakeEx(int ch, BSTR calledNum, BSTR callerNum, BSTR sURI, BSTR sContact, BSTR sAuthName, BSTR sAuthPassword, BSTR sDestIP, ushort nDestPort)

DLL: void GTAPI\_Send\_MakeEx(int ch, const char\* calledNum, const char\* callerNum, const char\* sURI, const char\* sContact, const char\* sAuthName, const char\* sAuthPassword, const char\* sDestIP, unsigned short nDestPort)

**Parameters:**

Ch: Channel Index

CalledNum: Called number address in SIP format. Must be string like:

"<sip:1234@abc.com>", "<sip:5678@10.98.1.1:5060>".

CallerNum: Caller id in SIP format. It can be "".

sURI: Request URI in SIP INVITE message. It can be ""

sContact: Local contact address. Leave it to "" for SDK to fill it.

sAuthName, sAuthPassword: for credit. SDK can choose a SIP account according to the caller and called string, but if you want to overwrite or specify a special username and password for the call, you call fill out these two parameters. Otherwise leave them "".

sDestIP, nDestPort: The IP address and port you want the INVITE message send to. Leave them "", 0 to let SDK decide.

**Return:**

null

**Sample code:**

//Make a call out on the first channel(0) to user 8888 on pcbest.net.

```
Send_MakeEx(0, "<sip:8888@pcbest.net>", "", "sip:8888@pcbest.net", "", "", "", "", 0);
```

### 3.2.3 Send\_Answer

**Description:** Answer an incoming call. The SDK will issue On\_RecvConnected event once the call is connected.

**Format:**

C++: void Send\_Answer(int ch);  
.NET: void Send\_Answer(int ch);  
OCX: void SendAnswer(int ch);  
DLL: void GTAPI\_Send\_Answer(int ch);

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//Auto Answer an incoming call
Void On_RecvOffer(in ch, ..... )
{
    Send_Answer(ch);
}
```

### 3.2.4 Send\_HungUp **deprecated (Use Send\_HangUp)**

**Description:** Hang up a connected call. The SDK will issue On\_RecvIdle event when the call disconnected successfully.

**Format:**

C++: void Send\_HungUp(int ch);  
.NET: void Send\_HungUp(int ch);  
OCX: void SendHungUp(int ch);  
DLL: void GTAPI\_Send\_HungUp(int ch);

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//DND(Do not disturb)
```

```

Void On_RecvOffer(in ch, ..... )
{
    Send_HungUp(ch);
}

```

### 3.2.5 Send\_HangUp

**Description:** Hang up a connected call or reject an incoming call which is not answered. The SDK will issue On\_RecvIdle event when the call disconnected successfully.

Support 302 server side, by using Send\_Hangup(Ex), so when a new call arrives, On\_RecvOffered(int ch, ....)

```

{
    Send_HangupEx(ch, 302, "Moved Moved Temporarily Contact:
<sip:newaddr@newdomain.com>");
}

```

**Format:**

C++: void Send\_HangUp(int ch, int code, const char\* desc);  
.NET: void Send\_HangUp(int ch, int code, string desc);  
OCX: void **SendHangUpEx**(int ch, int code, BSTR desc);  
DLL: void GTAPI\_Send\_HangUp(int ch, int code, const char\* desc);

**Parameters:**

Ch: Channel Index

Code: SIP Response Code. Most of time you should use 4xx code. Please refer to this page about SIP response code: <http://www.voip-info.org/wiki/view/SIP+Response+class4>

Desc: The description of SIP Response Code.

**Return:**

null

**Sample code:**

//Send DND(Do not disturb) or BUSY when receiving a new call

```

Void On_RecvOffer(in ch, ..... )
{
    Send_HungUp(ch, 486, "Busy Here");
}

```

### 3.2.6 Send\_Hold

**Description:** Hold or un-hold a call. The SDK will issue On\_RecvHolding event to tell if the hold operation succeed.

**Format:**

C++: void Send\_Hold(int ch);  
.NET: void Send\_Hold(int ch);  
OCX: void SendHold(int ch);  
DLL: void GTAPI\_Send\_Hold(int ch);

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//Hold the call on channel 0
Send_Hold(0);
```

If you have multiple lines, holding a call on one line and switching to another can be very complex, because you should maintain the your GUI buttons carefully. Here is a sample C++ code for Hold/Unhold. Please refer it when you writing hold/unhold code.

You need to follow the code to do hold and unhold:

- 1. define a structure to record channel's hold status. You can insert the following code into your channel structure if you already have.**

```
struct SIPPhone_Channel
{
    bool sent_hold; //if this channel has sent hold out
    int holding; //0=idle, 1=holding remote, 2=be held.
    SIPPhone_Channel()
    {
        sent_hold = false;
        holding = 0;
    }
};
```

- 2. Init the channel structure**

Assume you have 4 channels(lines)

```
SIPPhone_Channel m_pSIPchan = new SIPPhone_Channel[4]
```

- 3. Assume m\_chIndex is the current channel index that user is working on.**



```

void CSIPPhoneWnd::OnButtonHold() //when press hold button.
{
    CGTSIPPhoneEnv* penv = (CGTSIPPhoneEnv*)CGTAPIEnv::GetGTAPIEnv();
    m_pSIPChans[m_chIndex].sent_hold = true;
    penv->Send_Hold(m_chIndex);
}

void CSIPPhoneWnd::On_RecvHolding(int ch, int hold_on) //This is a SDK event
{
    CGTSIPPhoneEnv* penv = (CGTSIPPhoneEnv*)CGTAPIEnv::GetGTAPIEnv();
    GTAPI_Channel* pChan = penv->GetChannel(ch);
    if(!pChan) return;
    if(hold_on)
    {
        penv->Send_StopDXAudio(ch);
        if(m_pSIPChans[ch].sent_hold)
        {
            m_pSIPChans[ch].holding = 1; //holding remote
            if(ch == m_chIndex)
            {
                //user on current phone line
                //need to update the status of buttons
                m_btnDial.EnableWindow(FALSE);
                m_btnHungup.EnableWindow(TRUE);
                m_btnHold.EnableWindow(TRUE);
                m_btnTrans.EnableWindow(FALSE);
            }
            if(penv->m_bMOH) //if music on hold, start it.
            {
                penv->Send_StartMusicOnHold(ch, penv->m_sMOHFolder, 1, 0);
            }
        }
    }
    else
    {
        m_pSIPChans[ch].holding = 2; //be held
        if(ch == m_chIndex)
        {
            m_btnDial.EnableWindow(FALSE);
            m_btnHungup.EnableWindow(TRUE);
            m_btnHold.EnableWindow(FALSE);
            m_btnTrans.EnableWindow(FALSE);
        }
    }
}
else
{

```

```

if(m_pSIPChans[ch].sent_hold && penv->m_bMOH)
{
    penv->Send_StopMusicOnHold(ch);
}
m_pSIPChans[ch].sent_hold = false;
m_pSIPChans[ch].holding = 0;
penv->Send_StartDXAudio(ch);
if(ch == m_chIndex)
{
    //user working on this line
    m_btnDial.EnableWindow(FALSE);
    m_btnHungup.EnableWindow(TRUE);
    m_btnHold.EnableWindow(TRUE);
    m_btnTrans.EnableWindow(TRUE);
}
}
}

```

#### 4. This is the code when user switch lines:

```

void CSIPPhoneWnd::OnButtonLineSelected(unsigned int chIndex) //switch to chIndex
{
    CGTSIPPhoneEnv* penv = (CGTSIPPhoneEnv*)CGTAPIEnv::GetGTAPIEnv();
    m_chIndex = chIndex;

    for(unsigned int i=0; i < 4; i++) //go through all line buttons
    {
        if(i != chIndex) //if it is not the line
        {
            m_pSIPLines[i].m_btnLine.SetState(FALSE); //change the line button status or color
            to inactive
            if(pChan->ch_status == GTAPI_Channel::GTAPI_CHANNEL_IDLE) //if it is IDLE,
            update bitmap status
            SetLineButtonBitmap(i, 0);
        }
        else
        {
            m_pSIPLines[i].m_btnLine.SetState(TRUE); //change the line button to active status
            if(pChan->ch_status == GTAPI_Channel::GTAPI_CHANNEL_IDLE)
            {
                SetLineButtonBitmap(i, 1);
                SetSIPCallerOrCalledNumber("");
                m_btnDial.EnableWindow(TRUE);
                m_btnHungup.EnableWindow(FALSE);
                m_btnHold.EnableWindow(FALSE);
                m_btnTrans.EnableWindow(FALSE);
                LogLineStatus(i, "");
            }
        }
    }
}

```

```

}
else if(pChan->ch_status == GTAPI_Channel::GTAPI_CHANNEL_DIALNG || \
pChan->ch_status == GTAPI_Channel::GTAPI_CHANNEL_DISCONNECTING)
{
    SetLineButtonBitmap(i, 2);
    if(pChan->originate)
        SetSIPCallerOrCalledNumber(pChan->callee_num);
    else
        SetSIPCallerOrCalledNumber(pChan->caller_num);
    m_btnDial.EnableWindow(FALSE);
    m_btnHungup.EnableWindow(TRUE);
    m_btnHold.EnableWindow(FALSE);
    m_btnTrans.EnableWindow(FALSE);
}
else if(pChan->ch_status == GTAPI_Channel::GTAPI_CHANNEL_CONNECTED)
{
    SetLineButtonBitmap(i, 2);
    if(pChan->originate)
        SetSIPCallerOrCalledNumber(pChan->callee_num);
    else
        SetSIPCallerOrCalledNumber(pChan->caller_num);
    m_btnDial.EnableWindow(FALSE);
    m_btnHungup.EnableWindow(TRUE);
    switch(m_pSIPChans[i].holding) //check if this line is in holding, or be held
    {
        case 0:
            m_btnHold.EnableWindow(TRUE);
            m_btnTrans.EnableWindow(TRUE);
            break;
        case 1: //holding remote
            m_btnHold.EnableWindow(TRUE);
            m_btnTrans.EnableWindow(FALSE);
            break;
        case 2: //be holded by remote
            m_btnHold.EnableWindow(FALSE);
            m_btnTrans.EnableWindow(FALSE);
            break;
    }
}
else if(pChan->ch_status == GTAPI_Channel::GTAPI_CHANNEL_OFFERED)
{
    SetLineButtonBitmap(i, 2);
    if(pChan->originate)
        SetSIPCallerOrCalledNumber(pChan->callee_num);
    else
        SetSIPCallerOrCalledNumber(pChan->caller_num);
}

```

```

    m_btnDial.EnableWindow(TRUE);
    m_btnHungup.EnableWindow(TRUE);
    m_btnHold.EnableWindow(FALSE);
    m_btnTrans.EnableWindow(FALSE);
}

}
}
}

```

### 5. Finally don't forget to clear the line hold status in On\_RecvIdle and

#### On\_RecvConnected (actually only need to set in On\_RecvIdle)

```

void CSIPPhoneWnd::On_RecvConnected(GTAPI_Channel* ch)
{
    char sTemp[256] = "";
    // CGTSIPPhoneEnv* penv = (CGTSIPPhoneEnv*)CGTAPIEnv::GetGTAPIEnv();
    LogLineStatus(ch->ch_index, "Connected");
    if(ch->ch_index == m_chIndex)
    {
        m_btnDial.EnableWindow(FALSE);
        m_btnHungup.EnableWindow(TRUE);
        m_btnHold.EnableWindow(TRUE);
        m_btnTrans.EnableWindow(TRUE);
        // Not sure why sometimes the screen doesn't get freshed
        if(!*GetNumber(sTemp, 255))
        {
            if(ch->originate)
                SetSIPCallerOrCalledNumber(ch->callee_num);
            else
                SetSIPCallerOrCalledNumber(ch->caller_num);
        }
    }
    SetLineButtonBitmap(ch->ch_index, 2);

    m_pSIPChans[ch->ch_index].sent_hold = false;
    m_pSIPChans[ch->ch_index].holding = 0;
}

```

```

void CSIPPhoneWnd::On_RecvIdle(GTAPI_Channel* ch)
{
    // char sTemp[256];
    // memset(sTemp, 0, sizeof(sTemp));
    CGTSIPPhoneEnv* penv = (CGTSIPPhoneEnv*)CGTAPIEnv::GetGTAPIEnv();
    if(ch->originate && ch->ch_status !=
    GTAPI_Channel::GTAPI_CHANNEL_CONNECTED)
    {

```

```

    LogLineStatus(ch->ch_index, penv->GetChanLastMsgText(ch->ch_index));
}
else
    LogLineStatus(ch->ch_index, "Idle");
if(ch->ch_index == m_chIndex)
{
    SetSIPCallerOrCalledNumber("");
    m_btnDial.EnableWindow(TRUE);
    m_btnHungup.EnableWindow(FALSE);
    m_btnHold.EnableWindow(FALSE);
    m_btnTrans.EnableWindow(FALSE);
}
if(m_chIndex == ch->ch_index)
{
    SetLineButtonBitmap(ch->ch_index, 1);
}
else
{
    SetLineButtonBitmap(ch->ch_index, 0);
}
m_pSIPChans[ch->ch_index].sent_hold = false;
m_pSIPChans[ch->ch_index].holding = 0;
}

```

### 3.2.7 Send\_Transfer

**Description:** Transfer current active call to a new sip address. This is blind transfer method. Please make sure this channel is in an active call, and the call has been applied hold(used Send\_Hold).

**Format:**

C++: void Send\_Transfer(int ch, const char\* transTo);  
.NET: void Send\_Transfer(int ch, string transTo);  
OCX: void SendTransfer(int ch, BSTR transTo);  
DLL: void GTAPI\_Send\_Transfer(int ch, const char\* transTo);

**Parameters:**

Ch: Channel Index

transTo: The SIP address to transfer. It must be a SIP address like:

“<sip:username@abc.com>”

**Return:**

null

**Sample code:**

```
//Transfer the call on channel 0 to <sip:8888@pcbest.net>
Send_Transfer(0, "<sip:8888@pcbest.net>");
```

### 3.2.8 Send\_TransferEx

**Description:** Transfer current active call to a new sip address. This is attended transfer method. Please make sure this channel is in an active call, and the call has been applied hold(used Send\_Hold). Also the replace channel is connected, and in hold status.

**Format:**

```
C++: void Send_TransferEx(int ch, const char* transTo, int replace_ch);
.NET: void Send_TransferEx(int ch, string transTo, int replace_ch);
OCX: void SendTransferEx(int ch, BSTR transTo, int replace_ch);
DLL: void GTAPI_Send_TransferEx(int ch, const char* transTo, int replace_ch);
```

**Parameters:**

Ch: Channel Index. The channel has the original call, and need to be transferred to the transTo address.

transTo: The SIP address to transfer. It must be a SIP address like:

"<sip:username@abc.com>"

replace\_ch: The second channel that has been called transTo address.

**Return:**

null

**Sample code:**

```
//attended transfer steps
```

For example, channel ch has an active call, and you want to transfer the call to another address(transTo)

**Step 1:** Hold call on ch.

```
Send_Hold(ch)
```

**Step 2:** Received On\_RecvHolding event, indicate the call is holding. Then use an idle channel to call out transferee address.

```
void On_RecvHolding(int ch, int hold_on)
{
    if(ch is the channel which does attended transfer, and hold_on is 1)
```

```

{
    //You can use StopDXAudio(ch) to disable sound device if it is a softphone
    application.
    //Select an idle channel to call out to transTo.
    //Assume the ch1 is idle.
    Send_Make(ch1, transTo, "");
}
}

```

**Step 3:** In call connected event, hold the second call.

Void On\_RecvConnected(int ch)

```

{
    if(ch is ch1)
    {
        Send_Hold(ch1);
    }
}

```

**Step 4:** In call holding event, use Send\_TransferEx to do attended transfer.

void On\_RecvHolding(int ch, int hold\_on)

```

{
    If(ch is ch1)
    {
        Send_TransferEx(ch, transTo, ch1);
    }
}

```

### 3.2.9 Send\_Redirect

**Description:** Use 3xx response to redirect a call. You must set "gtsrv.sip.callcontrol.auto.ringcall" to 0, otherwise the SDK will automatically give "ring" signal back for new incoming calls.

**Format:**

C++: void Send\_Redirect(int ch, const char\* redirectTo, const char\* respCode, const char\* respStr);  
.NET: void Send\_Redirect(int ch, string redirectTo, string respCode, string respStr);  
OCX: void SendRedirect(int ch, BSTR redirectTo, BSTR respCode, BSTR respStr);  
DLL: void GTAPI\_Send\_Redirect(int ch, const char\* redirectTo, const char\* respCode, const char\* respStr);

**Parameters:**

Ch: Channel Index

redirectTo: The SIP address to redirect call to. It must be a SIP address like:

"<sip:username@abc.com>"

respCode: 3xx code. For example: "302".

respStr: 3xx response string. For example: "Moved Temporarily"

See the list following for all 3xx responses:

"300" "Multiple Choices"

"301" "Moved Permanently"

"302" "Moved Temporarily"

"305" "Use Proxy"

"380" "Alternative Service"

**Return:**

null

**Sample code:**

```
//Transfer the call on channel 0 to <sip:8888@pcbest.net>
```

```
Send_Redirect (0, "<sip:8888@pcbest.net>", "302", "Moved Temporarily");
```

### 3.2.10 Send\_Accept

**Description:** If "gtsrv.sip.callcontrol.auto.acceptcall" is set to 0, (default it is set to 1), this function is used to send remote side SIP TRY message. It means the SDK accepts the call.

**Format:**

C++: void Send\_Accept(int ch);

.NET: void Send\_Accept(int ch);

OCX: void SendAccept(int ch);

DLL: void GTAPI\_Send\_Accept(int ch);

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
Send_Accept (0);
```



### 3.2.11 Send\_Ring

**Description:** If "gtsrv.sip.callcontrol.auto.ringcall" is set to 0, (default it is set to 1), this function is used to send remote side SIP RING message. It means the SDK is ringing the local for the incoming call on channel ch.

**Format:**

C++: void Send\_Ring(int ch);  
.NET: void Send\_Ring(int ch);  
OCX: void SendRing(int ch);  
DLL: void GTAPI\_Send\_Ring(int ch);

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

Send\_Ring(0);

### 3.2.12 Send\_SessionProgress

**Description:** If you want to send 183 – session progress to an incoming call, which you haven't answered the call yet, you can invoke this function. By doing this, you can open the RTP port, and send remote end some tones, like ring tone, busy tone, or others. You need to prepare the tone file (sound file) yourself.

**Format:**

C++: void Send\_SessionProgress(int ch, const char\* fn);  
.NET: void Send\_SessionProgress(int ch, string fn);  
OCX: void Send\_SessionProgress(int ch, BSTR fn);  
DLL: void GTAPI\_Send\_SessionProgress(int ch, const char\* fn);

**Parameters:**

Ch: Channel Index  
fn: sound file name.

**Return:**

null

**Sample code:**

Send\_SessionProgress(0, "c:\\ringbacktone.wav");

### 3.2.13 Send\_WaitForCall (new function from 2.05f)

**Description:** Some developers prefer to allow inbound call only after calling a function WaitForCall. In default, SDK send the new incoming call into a channel when it is(turning into) IDLE. But if you prefer waitForCall mode to accept new call, you can set "gtsrv.wait.for.call.mode" to 1(default 0) before startServer, then call this function after channel is IDLE.

**Note:** if you use this mode and forget to call Send\_WaitForCall on the channel, this channel will **NOT** be able to accept any new incoming call until you called this function.

**Format:**

C++: void Send\_WaitForCall(int ch);  
.NET: void Send\_WaitForCall(int ch);  
OCX: void SendWaitForCall(LONG ch);  
DLL: void GTAPI\_Send\_WaitForCall(int ch);

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
Void On_RecvIdle(int ch) //channel turned idle
{
    Send_WaitForCall(ch); //call it to allow inbound call. ONLY when
    "gtsrv.wait.for.call.mode" is set to 1.
}
```

### 3.2.14 On\_RecvConnected

**Description:** Event to notify the call is connected.

**Format:**

C++: void On\_RecvConnected(int ch)  
.NET: void On\_RecvConnected(int ch)  
OCX: void OnCallConnected(int ch)  
DLL: void GTAPI\_SetCB\_On\_RecvConnected

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//Start DirectX audio on the connected channel, so this channel is working with
Sound device to present voice. This is the route for most softphone application.
Void On_RecvConnected(int ch)
{
    Send_StartDXAudio(ch);
}
```

### 3.2.15 On\_RecvOffered

**Description:** Event to notify a new incoming call. This event is only triggered for a new inbound call. Note, please override only one of this event and On\_RecvOfferedEx.

**Format:**

C++: void On\_RecvOffered(int ch, const char\* sCaller, const char\* sCallee, const char\* sDestAddr, const char\* sViaAddr, const char\* sFromIP, unsigned short nFromPort)

.NET: void On\_RecvOffered(int ch, string sCaller, string sCallee, string DestAddr, string sViaAddr, string sFromIP, unsigned short nFromPort)

OCX: void **OnCallOffered**(int ch, BSTR sCaller, BSTR sCallee, BSTR sDestAddr, BSTR sViaAddr, BSTR sFromIP, unsigned short nFromPort)

DLL: void GTAPI\_SetCB\_On\_RecvOffered

**Parameters:**

Ch: Channel Index

sCaller: Caller ID. It equals to SIP From field.

sCallee: Called ID. It equals to SIP To field.

sDestAddr: Call destination address. It equals to SIP Uri field.

sViaAddr: Call via address. It equals to the most top via address in SIP message.

sFromIP: the ip address that SIP message is from.

sFromPort: the ip port that SIP message is from.

**Return:**

null

**Sample code:**

```
//Softphone code. Play local ring tone in sound card to mention users that there is a
new incoming call
Void On_RecvOffered(int ch, ...)
{
    PlayLocalRingSound();
}
```

### 3.2.16 On\_RecvOfferedEx

**Description:** Event to notify a new incoming call. This event is only triggered for a new inbound call. Note, please override only one of this event and On\_RecvOffered.

**Format:**

C++: void On\_RecvOfferedEx(int ch, const char\* sCaller, const char\* sCallee, const char\* sDestAddr, const char\* sViaAddr, const char\* sFromIP, unsigned short nFromPort, const char\* sOrgSIPTxt, const char\* Reserved1, const char\* Reserved2, const char\* Reserved3)

.NET: void On\_RecvOfferedEx(int ch, string sCaller, string sCallee, string DestAddr, string sViaAddr, string sFromIP, unsigned short nFromPort, string sOrgSIPTxt, const string Reserved1, string Reserved2, string Reserved3)

OCX: void OnCallOfferedEx(int ch, BSTR sCaller, BSTR sCallee, BSTR sDestAddr, BSTR sViaAddr, BSTR sFromIP, unsigned short nFromPort, BSTR sOrgSIPTxt, BSTR Reserved1, BSTR Reserved2, BSTR Reserved3)

DLL: void GTAPI\_SetCB\_On\_RecvOfferedEx

**Parameters:**

Ch: Channel Index

sCaller: Caller ID. It equals to SIP From field.

sCallee: Called ID. It equals to SIP To field.

sDestAddr: Call destination address. It equals to SIP Uri field.

sViaAddr: Call via address. It equals to the most top via address in SIP message.

sFromIP: the ip address that SIP message is from.

sFromPort: the ip port that SIP message is from.

sOrgSIPTxt: the original SIP message in pure text

Reserved1, Reserved2, Reserved3: for future usage

**Return:**

null

**Sample code:**

```
//Softphone code. Play local ring tone in sound card to mention users that there is a
new incoming call
Void On_RecvOfferedEx(int ch, ...)
{
```

```

    PlayLocalRingSound();
}

```

### 3.2.17 On\_RecvDialing

**Description:** Event to notify that the channel is dialing out. This event is **only** triggered for outbound calls.

**Format:**

C++: void On\_RecvDialing(int ch, const char\* sCaller, const char\* sCallee)

.NET: void On\_RecvDialing(int ch, string sCaller, string sCallee)

OCX: void **OnCallDialing**(int ch, BSTR sCaller, BSTR sCallee)

DLL: void GTAPI\_SetCB\_On\_RecvDialing

**Parameters:**

Ch: Channel Index

sCaller: Caller ID. It equals to SIP From field.

sCallee: Called ID. It equals to SIP To field.

**Return:**

null

**Sample code:**

//Just printf the event

Void On\_RecvDialing (int ch, const char\* sCaller, const char\* sCallee)

```

{
    Printf("dialing out");
}

```

### 3.2.18 On\_RecvRinging

**Description:** Event to tell that the remote side ringed. This event is **only** triggered for outbound calls.

**Format:**

C++: void On\_RecvRinging(int ch)

.NET: void On\_RecvRinging(int ch)

OCX: void **OnCallRinging**(int ch)

DLL: void GTAPI\_SetCB\_On\_RecvRinging

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

//Softphone route. Play remote ring tone in local sound card.

Void On\_RecvRinging(int ch)

```
{
    PlayRemoteRingSound();
}
```

**3.2.19 On\_RecvIdle (New Format)**

**Description:** Event to tell that the call is end or the channel is idle, and ready to accept next inbound call or make outbound call.

**Format:**

C++: void On\_RecvIdle(int ch, int code, const char\* desc)

.NET: void On\_RecvIdle(int ch, int code, string desc)

OCX: void OnCallIdleEx(int ch, int code, BSTR desc)

DLL: void GTAPI\_SetCB\_On\_RecvIdle

**Parameters:**

Ch: Channel Index

Code: SIP Response Code. Please refer to this page about SIP response code:

<http://www.voip-info.org/wiki/view/SIP+Response+class4>

Desc: The description of SIP Response Code.

**Return:**

null

**Sample code:**

//Just log out when channel is idle

Void On\_RecvIdle(int ch, int code, const char\* desc)

```
{
    Printf("Channel is idle.");
}
```

### 3.2.20 On\_RecvSessionProgress

**Description:** This event indicates it received remote SIP ends 183 session progress message when making outbound calls. This event is only triggered when making calls out, and will carry info about the remote RTP audio. SDK will automatically open RTP media for this event, and you can record the audio on this channel for pre-connected audio tone, or use tone detecting method to analysis the ring tone.

**Format:**

C++: void On\_RecvSessionProgress(int ch)  
.NET: void On\_RecvSessionProgress (int ch)  
OCX: void OnCallSessionProgress(int ch)  
DLL: void GTAPI\_SetCB\_On\_RecvSessionProgress

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//Just log out
Void On_RecvSessionProgress (int ch)
{
    Printf("Session progress received.");
}
```

### 3.2.21 On\_RecvHolding

**Description:** Event to tell that the call is end or the channel is idle, and ready to accept next inbound call or make outbound call.

**Format:**

C++: void On\_RecvHolding(int ch, int hold\_on)  
.NET: void On\_RecvHolding(int ch, int hold\_on)  
OCX: void OnCallHolding(int ch, int hold\_on)  
DLL: void GTAPI\_SetCB\_On\_RecvHolding

**Parameters:**

Ch: Channel Index

Hold\_on: 1 = Call is holding, 0 = Call is unhold

**Return:**

null

**Sample code:**

```
//just log out
Void On_RecvHolding(int ch, int hold_on)
{
    If(hold_on == 1)
        Printf("Channel is holding");
}
```

*please refer to above Send\_Hold section for a complete Hold/Unhold code.*

### 3.2.22 On\_RecvTransferring

**Description:** Event to tell that the call is end or the channel is idle, and ready to accept next inbound call or make outbound call.

**Format:**

C++: void On\_RecvTransferring(int ch, const char\* sAddr, const char\* sReplaceCallID, const char\* sCallFromTag, const char\* sCallToTag)  
.NET: void On\_RecvTransferring(int ch, string sAddr, string sReplaceCallID, string sCallFromTag, string sCallToTag)  
OCX: void **OnCallTransferring**(int ch, BSTR sAddr, BSTR sReplaceCallID, BSTR sCallFromTag, BSTR sCallToTag)  
DLL: void GTAPI\_SetCB\_On\_RecvTransferring

**Parameters:**

Ch: Channel Index

sAddr: the dest address the call is transferring to

sReplaceCallID: The unique call id that needs to be replaced by call transferring.

sCallFromTag: The call from tag to indicate the call to be replaced

sCallToTag:: The call to tag to indicate the call to be replaced

**Return:**

null

**Sample code:**

```
//just log out
Void On_RecvTransferring (int ch, const char* sAddr, const char* sReplaceCallID,
const char* sCallFromTag, const char* sCallToTag)
{
    Printf("Call is transferring to %s", sAddr);
}
```



### 3.2.23 On\_RecvNoAudio

**Description:** Event to tell there is no incoming RTP audio for a period of time. You can use CFG\_SetValue("gtsrv.sip.no.audio.duration", "5000") to set how long you want the event to be triggered if there is no audio. This event is set here in call control session because it can be used to determine if a call is still alive. Sometimes if the remote end's Internet connection is down suddenly, you can use this event to terminate the call.

**Format:**

C++: void On\_RecvNoAudio(int ch, int reserved)  
.NET: void On\_RecvNoAudio(int ch, int reserved)  
OCX: void OnRecvNoAudio(int ch, int reserved)  
DLL: void GTAPI\_SetCB\_On\_RecvNoAudio

**Parameters:**

Ch: Channel Index  
reserved: not used now

**Return:**

null

**Sample code:**

```
//disconnect the call if received this event
Void On_RecvNoAudio (int ch, int reserved)
{
    Send_Hungup(ch);
}
```

### 3.2.24 GetChanCallID

**Description:** Get the call SIP unique ID. Only use this function when channel's status is not idle.

**Format:**

C++: const char\* GetChanCallID(int ch)  
.NET: string GetChanCallID(int ch)  
OCX: BSTR GetChanCallID(int ch)  
DLL: const char\* GTAPI\_GetChanCallID(int ch)

**Parameters:**

Ch: Channel Index

**Return:**

SIP unique ID to identify the call.

**Sample code:**

```
//Get the first channel's call id.
const char *sID = env.GetChanCallID(0);
```

### 3.2.25 GetChanCallFromTag

**Description:** Get the call from tag which identifies the call. Only use this function when channel's status is not idle.

**Format:**

```
C++: const char* GetChanCallFromTag(int ch)
.NET: string GetChanCallFromTag(int ch)
OCX: BSTR GetChanCallFromTag(int ch)
DLL: const char* GTAPI_GetChanCallFromTag(int ch)
```

**Parameters:**

Ch: Channel Index

**Return:**

SIP Call From-Tag to identify the call.

**Sample code:**

```
//Get the first channel's from-tag.
const char *sID = env. GetChanCallFromTag (0);
```

### 3.2.26 GetChanCallToTag

**Description:** Get the call to tag which identifies the call. Only use this function when channel's status is not idle.

**Format:**

```
C++: const char* GetChanCallToTag(int ch)
.NET: string GetChanCallToTag(int ch)
OCX: BSTR GetChanCallToTag(int ch)
```

DLL: `const char* GTAPI_GetChanCallToTag(int ch)`

**Parameters:**

Ch: Channel Index

**Return:**

SIP Call To-Tag to identify the call.

**Sample code:**

```
//Get the first channel's to-tag.
const char *sID = env.GetChanCallToTag (0);
```

### 3.3 Channel Status Functions

There are different ways to access channel's status.

**C++ and .NET:** use `GetChannel` to get `GTAPI_Channel*` pointer. Then use `ch_status` member to access the status of the channel. Status can be:

```
0 = GTAPI_CHANNEL_IDLE,
1 = GTAPI_CHANNEL_OFFERED,
2 = GTAPI_CHANNEL_DIALNG,
3 = GTAPI_CHANNEL_CONNECTED,
4 = GTAPI_CHANNEL_DISCONNECTING,
5 = GTAPI_CHANNEL_RESERVED
```

**OCX and DLL:** use `Send_GetChanStatus` to trigger the event `On_RecvStatus`.

#### 3.3.1 Send\_GetChanStatus

**Description:** Get the channel's status. An event `On_RecvStatus` will be triggered later.

**Format:**

```
C++: void Send_GetChanStatus(int ch)
.NET: void Send_GetChanStatus(int ch)
OCX: void SendGetChanStatus(int ch);
DLL: void GTAPI_Send_GetChanStatus(int ch)
```

**Parameters:**

ch: Channel index(based on 0).

**Return:**

null

**Sample code:**

```
//get the channel status
env.Send_GetChannelStatus(0);
```

### 3.3.2 On\_RecvStatus

**Description:** Event to the channel's status.

**Format:**

C++: void On\_RecvStatus(int chBegin, int chEnd, int chStatus)

.NET: void On\_RecvStatus(int chBegin, int chEnd, int chStatus)

OCX: void OnRecvStatus(int chBegin, int chEnd, int chStatus)

DLL: void GTAPI\_SetCB\_On\_RecvStatus

**Parameters:**

chBegin and chEnd: Same. Channel Index.

chStatus: status code.

//Channel status could be:

// 0 = GT\_CALL\_IDLE,

// 1 = GT\_CALL\_DIALING,

// 2 = GT\_CALL\_RINGING,

// 3 = GT\_CALL\_OFFERED,

// 4 = GT\_CALL\_CONNECTING,

// 5 = GT\_CALL\_CONNECTED,

// 6 = GT\_CALL\_DISCONNECTING,

// 7 = GT\_CALL\_RELEASING,

// 8 = GT\_CALL\_NOT\_AVAILABLE, //When span is not up

// 9 = GT\_CALL\_OFF\_LINE, // when resource is limited

// 10 = GT\_CALL\_MARKED\_BUSY, //when manually marked busy or maint

// 11 = GT\_CALL\_RESERVED, //reserved

**Return:**

null

**Sample code:**

//just log out

Void On\_RecvStatus(int chBegin, int chEnd, int chStatus)

{

    Printf("channel %d status %d", chBegin, chStatus);

}

### 3.3.3 Send\_SetChanStatus

**Description:** Set the channel's status. It can be used to disable a channel for a while(no incoming call on this channel).

**Format:**

C++: void Send\_SetChanStatus(int ch, int status)  
.NET: void Send\_SetChanStatus(int ch, int status)  
OCX: void SendSetChanStatus(int ch, int status)  
DLL: void GTAPI\_Send\_SetChanStatus(int ch, int status)

**Parameters:**

ch: Channel index(based on 0).

status:

//Channel status could be:  
// 0 = GT\_CALL\_IDLE,  
// 1 = GT\_CALL\_DIALING,  
// 2 = GT\_CALL\_RINGING,  
// 3 = GT\_CALL\_OFFERED,  
// 4 = GT\_CALL\_CONNECTING,  
// 5 = GT\_CALL\_CONNECTED,  
// 6 = GT\_CALL\_DISCONNECTING,  
// 7 = GT\_CALL\_RELEASING,  
// 8 = GT\_CALL\_NOT\_AVAILABLE, //When span is not up  
// 9 = GT\_CALL\_OFF\_LINE, // when resource is limited  
// 10 = GT\_CALL\_MARKED\_BUSY, //when manually marked busy or maint  
// 11 = GT\_CALL\_RESERVED, //reserved  
// 12 = GT\_CALL\_WAIT //wait for call status, when only  
"gtsrv.wait.for.call.mode" is set to 1

**Return:**

null

**Sample code:**

```
//reserve channel 0, so it cannot accept incoming calls
env.Send_SetChanStatus(0, 11);
```

### 3.3.4 SetChanDir (New function since 2.05f)

**Description:** Set the channel's call direction.

**Format:**

C++: void SetChanDir (int ch, int dir)

.NET: void SetChanDir (int ch, int dir)  
 OCX: void SetChanDir (int ch, int dir)  
 DLL: void GTAPI\_SetChanDir(int ch, int dir)

**Parameters:**

ch: Channel index(based on 0).

dir:

- 1 = Accept inbound call only,
- 2 = Make outbound call only,
- 3 = Both inbound and outbound(default),

**Return:**

null

**Sample code:**

```
//set channel 0,1 do inbound, and 2,3 do outbound
env.SetChanDir(0, 1);
env.SetChanDir(1, 1);
env.SetChanDir(2, 2);
env.SetChanDir(3, 2);
```

### 3.3.5 SetChanAudioDir

**Description:** Force channel's audio direction when negotiating with remote side. Typically, receive only audio for the call will send a=recvonly in SDP of INVITE message,..... Send only will present a=sendonly in SDP. This function is per call function so you have to set it before making a call or answering a call.

**Format:**

C++: void SetChanAudioDir (int ch, int dir)  
 .NET: void SetChanAudioDir (int ch, int dir)  
 OCX: void SetChanAudioDir (int ch, int dir)  
 DLL: void GTAPI\_SetChanAudioDir(int ch, int dir)

**Parameters:**

ch: Channel index(based on 0).

dir:

- 0 = No audio path/No media
- 1 = Send audio only
- 2 = Receive Audio only
- 3 = Both

**Return:**

null

## 3.4 SIP Accounts Status Functions

### 3.4.1 Set SIP Accounts

The SIP account information need to be preset into the code by CFG\_SetValue. In order to set the SIP accounts, you need the following tags:

```

“gtsrv.sip.reg.client.num” //how many sip accounts you have

“gtsrv.sip.reg1.displayname” //display name of account 1
“gtsrv.sip.reg1.username” //user name of account 1
“gtsrv.sip.reg1.domain” //SIP domain name of account 1
“gtsrv.sip.reg1.proxy” //SIP Proxy name of account 1
“gtsrv.sip.reg1.authorization” //Authorization name. In most cases, it equals to username.
“gtsrv.sip.reg1.password” //Password of account 1
“gtsrv.sip.reg1.expire” //How many seconds to register on the server
“gtsrv.sip.reg1.register” //1 = register on the server to receive incoming calls, 0=no
“gtsrv.sip.reg1.prot” //0 = UDP(default), 1 = TCP
“gtsrv.sip.reg1.userid” //Add special ";user=" into contact when registering on
SIP server, so when calling back, this id is in the URI that receiver can know who this
call is for
“gtsrv.sip.reg1.max.calls” //maximum calls of this account which can be dialed
“gtsrv.sip.reg1.retry.interval” //retry interval for failed registration. default 10 seconds

```

```

“gtsrv.sip.reg2.displayname” ... //if there is account 2
.....

```

Sample code to set SIP accounts(two accounts) in C++:

```

CFG_SetValue("gtsrv.sip.reg.client.num", "2");

CFG_SetValue("gtsrv.sip.reg1.displayname", "Any1");
CFG_SetValue("gtsrv.sip.reg1.username", "1234");
CFG_SetValue("gtsrv.sip.reg1.domain", "pcbest.net");
CFG_SetValue("gtsrv.sip.reg1.proxy", "pcbest.net");
CFG_SetValue("gtsrv.sip.reg1.authorization", "1234");
CFG_SetValue("gtsrv.sip.reg1.password", "xxxxxx");
CFG_SetValue("gtsrv.sip.reg1.expire", "3600");

CFG_SetValue("gtsrv.sip.reg2.displayname", "Any2");
CFG_SetValue("gtsrv.sip.reg2.username", "4567");
CFG_SetValue("gtsrv.sip.reg2.domain", "10.98.1.10");
CFG_SetValue("gtsrv.sip.reg2.proxy", "10.98.1.10");
CFG_SetValue("gtsrv.sip.reg2.authorization", "4567");

```

```
CFG_SetValue("gtsrv.sip.reg2.password", "xxxxxxx");
CFG_SetValue("gtsrv.sip.reg2.expire", "3600");
```

### 3.4.2 SIP Accounts Register Event(On\_RecvRegStatus)

**Description:** Event to tell the status of SIP accounts. This event can be triggered by function Send\_GetRegStatus.

**Format:**

```
C++: void On_RecvRegStatus(int user_id, int status, int regtime)
.NET: void On_RecvRegStatus(int user_id, int status, int regtime)
OCX: void OnRecvRegStatus(int user_id, int status, int regtime)
DLL: void GTAPI_SetCB_On_RecvRegStatus
```

**Parameters:**

user\_id: sip account id. From 0. for example, 0 means sip account "grsrv.sip.reg1.xxx.xxx" account, 1 equals to sip account "gtsrv.sip.reg2.xxx.xxx".  
 status: the sip account's status. 1 = successfully registered on the sip server. 0=no  
 regtime: in seconds. How many seconds are registered on the SIP server.

**Return:**

null

**Sample code:**

```
//just log out
Void On_RecvRegStatus(int user_id, int status, int regtime)
{
    If(status == 1)
        Printf("registered");
    Else
        Printf("not registered yet!");
}
```

### 3.4.3 Send\_GetRegStatus

**Description:** Use this function to trigger a SIP account event On\_RecvRegStatus.

**Format:**

```
C++: void Send_GetRegStatus(int user_id)
.NET: void Send_GetRegStatus(int user_id)
OCX: void SendGetRegStatus(int user_id)
DLL: void GTAPI_Send_GetRegStatus(int user_id)
```



**Parameters:**

user\_id: sip account id. From 0. for example, 0 means sip account “grsrv.sip.reg1.xxx.xxx” account, 1 equals to sip account “gtsrv.sip.reg2.xxx.xxx”.

**Return:**

null

**Sample code:**

```
//get the first account information after startServer
env.StartServer();
env.Send_GetRegStatus(0);
```

### 3.4.4 SIPAccount\_Register

**Description:** Call this function to register and unregister a sip account explicitly.

**Format:**

C++: void SIPAccount\_Register(int idx, bool b)  
.NET: void SIPAccount\_Register(int idx, bool b)  
OCX: void SIPAccount\_Register(int idx, bool b)  
DLL: void GTAPI\_SIPAccount\_Register(int idx, bool b)

**Parameters:**

idx: sip account id. From 0. for example, 0 means sip account “grsrv.sip.reg1.xxx.xxx” account, 1 equals to sip account “gtsrv.sip.reg2.xxx.xxx”.  
b: true = do register, false = do unregister

**Return:**

null

## 3.5 Dynamical SIP Account Management (New Feature of SDK v1.71f)

From the version v1.71f, you can dynamically add and remove SIP accounts for your application.

### 3.5.1 SIPAccount\_Add

**Description:** Add a SIP account.

**Format:**

C++: unsigned long SIPAccount\_Add(const char\* dp\_name, const char\* username, const char \*domain, const char\* proxy, const char\* authorization, const char \*password, int expire, int bReg, int maxSimultaneousCalls, int retryInterval, int bEnabled, int prot)

.NET: ulong SIPAccount\_Add(string dp\_name, string username, string domain, string proxy, string authorization, string password, int expire, int bReg, int maxSimultaneousCalls, int retryInterval, int bEnabled, int prot)

OCX: long SIPAccount\_Add(BSTR dp\_name, BSTR username, BSTR domain, BSTR proxy, BSTR authorization, BSTR password, int expire, int bReg, int maxSimultaneousCalls, int retryInterval, int bEnabled, int prot)

DLL: unsigned long GTAPI\_SIPAccount\_Add(const char\* dp\_name, const char\* username, const char \*domain, const char\* proxy, const char\* authorization, const char \*password, int expire, int bReg, int maxSimultaneousCalls, int retryInterval, int bEnabled, int prot)

**Parameters:**

dp\_name: Display name of the SIP account.

username: User name of the SIP account

domain: Domain name of the SIP account

proxy: Proxy name of the SIP account, usually same as domain

authorization: Username for authorization, usually same as username above

password: Password of SIP account

expire: Registration expiration time in seconds

bReg: If register to the SIP server to accept incoming call. 1 = register 0 = not register

maxSimultaneousCalls: 0 = unlimited. Otherwise it is limited.

retryInterval: 0 = default. Otherwise please set the period in seconds.

bEnabled: 1 = enabled, 0 = disabled

prot: 0 = UDP(default), 1 = TCP

**Return:**

SIP Account Handle. It can be used later for removing, or accessing info.

**Sample code:**

```
unsigned long accHandle = env.SIPAccount_Add("Myname", "87654321",
"pcbest.net", "pcbest.net", "87654321", "123456", 3600, 1, 0, 0, 1, 0);
```

### 3.5.2 SIPAccount\_Remove

**Description:** Remove a SIP account.

**Format:**

C++: void SIPAccount\_Remove(unsigned long h);  
.NET: void SIPAccount\_Remove(unsigned long h);  
OCX: void SIPAccount\_Remove(long h);  
DLL: void GTAPI\_SIPAccount\_Remove(unsigned long h);

**Parameters:**

h: Account handle returned by SIPAccount\_Add.

**Return:**

null

**Sample code:**

```
env.SIPAccount_Remove(accHandle);
```

### 3.5.3 SIPAccount\_Count

**Description:** Get the count of SIP accounts in the system.

**Format:**

C++: int SIPAccount\_Count();  
.NET: int SIPAccount\_Count();  
OCX: int SIPAccount\_Count();  
DLL: int GTAPI\_SIPAccount\_Count();

**Parameters:**

null

**Return:**

null

**Sample code:**

```
int num = env.SIPAccount_Count();
```

### 3.5.4 SIPAccount\_Index

**Description:** Get the index of SIP account handle

**Format:**

C++: int SIPAccount\_Index(unsigned long h);  
.NET: int SIPAccount\_Index(unsigned long h);  
OCX: int SIPAccount\_Index(long h);  
DLL: int GTAPI\_SIPAccount\_Index(unsigned long h);

**Parameters:**

h: SIP account handle which is returned by SIPAccount\_Add

**Return:**

The index of SIP account handle, based on 0.

**Sample code:**

```
int idx = env.SIPAccount_Index (h);
```

### 3.5.5 SIPAccount\_Handle

**Description:** Get the SIP account handle from index

**Format:**

C++: unsigned long SIPAccount\_Handle(int idx);  
.NET: unsigned long SIPAccount\_Handle(int idx);  
OCX: long SIPAccount\_Handle(int idx);  
DLL: unsigned long GTAPI\_SIPAccount\_Handle(int idx);

**Parameters:**

idx: the index of sip account

**Return:**

The handle of SIP account. 0 if the idx is invalid.

**Sample code:**

```
unsigned long = env.SIPAccount_Index (0); //get the first account's handle
```

### 3.5.6 SIPAccount\_Get

**Description:** Get the SIP account info from index

**Format:**

C++: const char\* SIPAccount\_Get(int idx, int type);  
.NET: string SIPAccount\_Get(int idx, int type);  
OCX: BSTR SIPAccount\_Get(int idx, int type);  
DLL: const char\* SIPAccount\_Get(int idx, int type);

**Parameters:**

idx: the index of sip account  
 type: What type of info you want to get.  
 0 = display name  
 1 = username  
 2 = domain  
 3 = proxy  
 4 = authorization  
 5 = password

**Return:**

The string of info

**Sample code:**

```
const char* domain_name = env.SIPAccount_Get(0, 2); //get the first account's
domain
```

### 3.5.7 SIPAccount\_Enable

**Description:** Enable/Disable a SIP account.

**Format:**

C++: void SIPAccount\_Enable(GT\_HANDLE h, bool b);  
 .NET: void SIPAccount\_Enable(uint32 h, bool b)  
 OCX: void SIPAccount\_Enable(long h, bool b);  
 DLL: const char\* GTAPI\_SIPAccount\_Enable(GT\_HANDLE h, bool b);

**Parameters:**

h: the handle of sip account  
 b: Boolean to enable or disable sip account

**Return:**

null

**Sample code:**

## 3.6 DTMF Functions and Events

You can use "gtsrv.sip.dtmf.method" to set DTMF type you want to use globally before calling StartServer function. In default, this value is 3, means Auto(Inband or RTP).

0 = inband, 1=SIP INFO, 2=RTP(RFC 2833), 3=Auto(Inband or RTP)

Samples:

```
CFG_SetValue("gtsrv.sip.dtmf.method", "1"); //Set DTMF type as SIP INFO.
```

### 3.6.1 Send\_PlayDTMFStr

**Description:** Play DTMF string to remote side, just like the user pressed buttons on the phone. This function is used for softphone application to send remote side DTMF keys.

**Format:**

```
C++: void Send_PlayDTMFStr(int ch, const char* str)
.NET: void Send_PlayDTMFStr(int ch, string str)
OCX: void SendPlayDTMFStr(int ch, BSTR str)
DLL: void GTAPI_Send_PlayDTMFStr(int ch, const char* str)
```

**Parameters:**

Ch: Channel Index  
Str: DTMF string

**Return:**

null

**Sample code:**

```
//Sending DTMF "1" when number button 1 is clicked
Void On_button_1_clicked()
{
    Env.SendDTMFStr(0, "1");
}
```

### 3.6.2 Send\_EnableDTMF

**Description:** Enable DTMF detection. This function is used for server application to start DTMF detection.

**Format:**

```
C++: void Send_EnableDTMF(int ch, int iMaxDigit, const char* termStr, int iMaxTimer)
.NET: void Send_EnableDTMF(int ch, int iMaxDigit, string termStr, int iMaxTimer)
OCX: void SendEnableDTMF(int ch, int iMaxDigit, BSTR termStr, int iMaxTimer)
DLL: void GTAPI_Send_EnableDTMF(int ch, int iMaxDigit, const char* termStr, int iMaxTimer)
```

**Parameters:**

Ch: Channel Index

iMaxDigit: Max digits to receive. 0 = unlimited

termStr: Stop DTMF detection once key in termStr detected. "" = no termStr

iMaxTimer: in **milliseconds**. timeout for DTMF detection. 0 = no timeout**Return:**

null

**Sample code:**

```
//Start DTMF detection when # is pressed or 4 digits received
Env.Send_EnableDTMF(0, 4, "#", 0);
```

### 3.6.3 Send\_DisableDTMF

**Description:** Disable DTMF detection. This function is used for server application to stop DTMF detection.

**Format:**

C++: void Send\_DisableDTMF(int ch)

.NET: void Send\_DisableDTMF(int ch)

OCX: void SendDisableDTMF(int ch)

DLL: void GTAPI\_Send\_DisableDTMF(int ch)

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//Stop DTMF detection
Env.Send_DisableDTMF(0);
```

### 3.6.4 On\_RecvDTMFDone

**Description:** Event to indicate the DTMF detection condition meets, and is done.

**Format:**

C++: void On\_RecvDTMFDone(int ch, int reason, const char\* dtmfBuf)

.NET: void On\_RecvDTMFDone(int ch, int reason, string dtmfBuf)

OCX: void OnRecvDTMFDone(int ch, int reason, BSTR dtmfBuf)  
 DLL: void GTAPI\_SetCB\_On\_RecvDTMFDone

**Parameters:**

Ch: Channel Index

Reason: DTMF Done reason could be:

// 0 = DTMF\_DONE\_TIMEOUT,

// 1 = DTMF\_DONE\_MAX\_DIGITS,

// 2 = DTMF\_DONE\_TERM\_DIGIT\_DETECTED,

dtmfBuf: DTMF string received.

**Return:**

null

**Sample code:**

//Just logout

```
Void On_RecvDTMFDone(int ch, int reason, const char* dtmfBuf)
{
    Printf("DTMF is done");
}
```

### 3.6.5 On\_RecvDTMFKeyDown

**Description:** Event to indicate that a DTMF key is pressed by remote side.

**Format:**

C++: void On\_RecvDTMFKeyDown(int ch, unsigned char keyValue, unsigned long ticks)

.NET: void On\_RecvDTMFKeyDown(int ch, unsigned char keyValue, unsigned long ticks)

OCX: void OnRecvDTMFKeyDown(int ch, unsigned char keyValue, unsigned long ticks)

DLL: void GTAPI\_SetCB\_On\_RecvDTMFKeyDown

**Parameters:**

Ch: Channel Index

keyValue: the key to be pressed by far-end

ticks: time of ticks

**Return:**

null

**Sample code:**

//Just logout



```

Void On_RecvDTMFKeyDown(int ch, unsigned char keyValue, unsigned long ticks)
{
    Printf("%c is pressed", keyValue);
}

```

### 3.6.6 On\_RecvDTMFKeyUp

**Description:** Event to indicate that a DTMF key is released by remote side.

**Format:**

```

C++: void On_RecvDTMFKeyUp(int ch, unsigned char keyValue, unsigned long ticks)
.NET: void On_RecvDTMFKeyUp (int ch, unsigned char keyValue, unsigned long ticks)
OCX: void OnRecvDTMFKeyUp(int ch, unsigned char keyValue, unsigned long ticks)
DLL: void GTAPI_SetCB_On_RecvDTMFKeyUp

```

**Parameters:**

Ch: Channel Index  
keyValue: the key to be released by far-end  
ticks: time of ticks

**Return:**

null

**Sample code:**

```

//Just log info out
Void On_RecvDTMFKeyUp(int ch, unsigned char keyValue, unsigned long ticks)
{
    Printf("%c is released", keyValue);
}

```

### 3.6.7 SetChanDTMFType

**Description:** Instead of using "gtsrv.sip.dtmf.method", you can set DTMF type for each channel in runtime.

**Format:**

```

C++: void SetChanDTMFType(int ch, unsigned int dtmfType);
.NET: void SetChanDTMFType(int ch, uint dtmfType);
OCX: void SetChanDTMFType(int ch, unsigned int dtmfType);

```

DLL: void GTAPI\_SetChanDTMFType(int ch, unsigned int dtmfType);

**Parameters:**

Ch: Channel Index

dtmfType: 0 = inband, 1=SIP INFO, 2=RTP(RFC 2833), 3=Auto(Inband or RTP)

**Return:**

null

**Sample code:**

```
SetChanDTMFType(0, 1); //Set the first channel uses SIP INFO to pass DTMF
```

### 3.6.8 GetChanDTMFType

**Description:** Get channel's DTMF type in runtime.

**Format:**

C++: unsigned int GetChanDTMFType(int ch, unsigned int dtmfType);

.NET: uint GetChanDTMFType(int ch, uint dtmfType);

OCX: unsigned int GetChanDTMFType(int ch, unsigned int dtmfType);

DLL: unsigned int GTAPI\_GetChanDTMFType(int ch, unsigned int dtmfType);

**Parameters:**

Ch: Channel Index

**Return:**

0 = inband, 1=SIP INFO, 2=RTP(RFC 2833), 3=Auto(Inband or RTP)

## 3.7 CT\_BUS Functions

CT\_BUS functions are used to connect two active channels' media. There are two kinds of connections. One is full duplex connection, another one is half duplex connect.

**Full Duplex: (A and B)**

A out => B in

B out => A in

**Half Duplex: (A and B)**

A out => B in

So from the above list, you probably will realize FullDuplexConnect(A and B) equals to HalfDuplexConnect(A and B) and HalfDuplexConnect(B and A).

Also echo test on channel A can be implemented by doing this:  
HalfDuplexConnect(A and A)

### 3.7.1 Send\_DuplexConnect

**Description:** Full Duplex Connect two channels. FullDuplexConnect can be used to connect two channels even with different audio codec. For example, channel A is working GSM, but channel B is working G711, but you can still connect two channels and make them talk with each other.

**Format:**

C++: void Send\_DuplexConnect(int ch1, int ch2)  
.NET: void Send\_DuplexConnect(int ch1, int ch2)  
OCX: void SendDuplexConnect(int ch1, int ch2)  
DLL: void GTAPI\_Send\_DuplexConnect(int ch1, int ch2)

**Parameters:**

Ch1: First Channel Index  
Ch2: Second Channel Index

**Return:**

null

**Sample code:**

```
//Full duplex connect channel 0 and 1
env.Send_DuplexConnect(0, 1);
```

### 3.7.2 Send\_DuplexDisconnect

**Description:** Full Duplex Disconnect two channels.

**Format:**

C++: void Send\_DuplexDisconnect(int ch1, int ch2)  
.NET: void Send\_DuplexDisconnect(int ch1, int ch2)  
OCX: void SendDuplexDisconnect(int ch1, int ch2)  
DLL: void GTAPI\_Send\_DuplexDisconnect(int ch1, int ch2)

**Parameters:**

Ch1: First Channel Index  
 Ch2: Second Channel Index

**Return:**  
 null

**Sample code:**  

```
//Full duplex disconnect channel 0 and 1
env.Send_DuplexDisconnect(0, 1);
```

### 3.7.3 Send\_HalfConnect

**Description:** Half Duplex Connect two channels.

**Format:**  
 C++: void Send\_HalfConnect(int ch1, int ch2)  
 .NET: void Send\_HalfConnect(int ch1, int ch2)  
 OCX: void SendHalfConnect(int ch1, int ch2)  
 DLL: void GTAPI\_Send\_HalfConnect(int ch1, int ch2)

**Parameters:**  
 Ch1: First Channel Index  
 Ch2: Second Channel Index

**Return:**  
 null

**Sample code:**  

```
//Half duplex connect channel 0 and 1
env.Send_HalfConnect(0, 1);
```

### 3.7.4 Send\_HalfDisconnect

**Description:** Half Duplex Disconnect two channels.

**Format:**  
 C++: void Send\_HalfDisconnect(int ch1, int ch2)  
 .NET: void Send\_HalfDisconnect(int ch1, int ch2)  
 OCX: void SendHalfDisconnect(int ch1, int ch2)  
 DLL: void GTAPI\_Send\_HalfDisconnect(int ch1, int ch2)

**Parameters:**

Ch1: First Channel Index  
 Ch2: Second Channel Index

**Return:**  
 null

**Sample code:**  

```
//Half duplex disconnect channel 0 and 1
env.Send_HalfDisconnect (0, 1);
```

### 3.7.5 Send\_RTPDuplexConnect

**Description:** Full duplex connect two channels by just forwarding two channels RTP packages. RTPDuplexConnect can be used to connect two channels which have the same audio codec, and you don't want the SDK to spend CPU time to deal with audio encoding and decoding.

**Format:**

C++: void Send\_RTPDuplexConnect(int ch1, int ch2)  
 .NET: void Send\_RTPDuplexConnect(int ch1, int ch2)  
 OCX: void SendRTPDuplexConnect(int ch1, int ch2)  
 DLL: void GTAPI\_Send\_RTPDuplexConnect(int ch1, int ch2)

**Parameters:**

Ch1: First Channel Index  
 Ch2: Second Channel Index

**Return:**  
 null

**Sample code:**  

```
//RTP full duplex connect channel 0 and 1
env.Send_RTPDuplexConnect (0, 1);
```

### 3.7.6 Send\_RTPDuplexDisconnect

**Description:** Disconnect two channels' RTP connect.

**Format:**

C++: void Send\_RTPDuplexDisconnect(int ch1, int ch2)  
 .NET: void Send\_RTPDuplexDisconnect (int ch1, int ch2)  
 OCX: void SendRTPDuplexDisconnect(int ch1, int ch2)  
 DLL: void GTAPI\_Send\_RTPDuplexDisconnect(int ch1, int ch2)

**Parameters:**

Ch1: First Channel Index

Ch2: Second Channel Index

**Return:**

null

**Sample code:**

```
//RTP duplex disconnect channel 0 and 1
env.Send_RTDP duplexDisconnect (0, 1);
```

### 3.7.7 Send\_ProxyConnect and Send\_ProxyDisconnect

**Description:** Proxy connect two channels. When two channels are proxy connected, the actual media(audio) is bypassed between 2 far ends.

**Format:**

C++: void Send\_ProxyConnect (int ch1, int ch2)

.NET: void Send\_ProxyConnect (int ch1, int ch2)

OCX: void SendProxyConnect (int ch1, int ch2)

DLL: void GTAPI\_Send\_ProxyConnect (int ch1, int ch2)

**Parameters:**

Ch1: First Channel Index

Ch2: Second Channel Index

**Return:**

null

**Sample code:**

```
env.Send_ProxyConnect (0, 1);
```

## 3.8 Audio Control Functions and Events

Audio control functions are used to play or record audio files on the active channels. Many times you may need to give a prompt in a live call, or record remote side's voice. These functions will assist your development by integrating the DTMF detection into playing and recording audio.

### 3.8.1 Send\_PlayAudio

**Description:** Play a sound file on the channel. Only channels in connected status can play a sound file. It supports DTMF detection conditions. Once the DTMF condition reached, the sound will stop playing, and On\_RecvAudioPlayDone event will be triggered to indicate which DTMF condition meets.

**Format:**

C++: void Send\_PlayAudio(int ch, const char\* audioFileName, int iMaxDigit, const char\* termStr, int iMaxTimer, unsigned int uStartByte)

.NET: void Send\_PlayAudio(int ch, string audioFileName, int iMaxDigit, string termStr, int iMaxTimer, unsigned int uStartByte)

OCX: void SendPlayAudioEx(int ch, BSTR audioFileName, int iMaxDigit, BSTR termStr, int iMaxTimer, unsigned int uStartByte);

DLL: void GTAPI\_Send\_PlayAudio

**Parameters:**

Ch: Channel Index

audioFileName: the sound file name

iMaxDigit: DTMF condition. 0=unlimited

termStr: DTMF detection condition.

iMaxTimer: in **milliseconds**. DTMF detection condition for timeout. 0 = no timeout

uStartByte: offset to start play audio in the sound file.

**Return:**

null

**Sample code:**

//Play a sound file

env.Send\_PlayAudio(0, "c:\abc.wav", 0, "", 0, 0);

### 3.8.2 Send\_RecordAudio

**Description:** Record **incoming only** voice into a sound file on the channel. Only channels in connected status can record audio. It supports DTMF detection conditions. Once the DTMF condition reached, the recording will stop, and On\_RecvAudioRecordDone event will be triggered to indicate which DTMF condition meets.

**Format:**

C++: void Send\_RecordAudio(int ch, const char\* audioFileName, int iMaxDigit, const char\* termStr, int iMaxTimer, unsigned int uStartByte)

.NET: void Send\_RecordAudio(int ch, string audioFileName, int iMaxDigit, string termStr, int iMaxTimer, unsigned int uStartByte)  
 OCX: void SendRecordAudioEx(int ch, BSTR audioFileName, int iMaxDigit, BSTR termStr, int iMaxTimer, unsigned int uStartByte)  
 DLL: void GTAPI\_Send\_RecordAudio

**Parameters:**

Ch: Channel Index

audioFileName: the sound file name

iMaxDigit: DTMF condition. 0=unlimited

termStr: DTMF detection condition.

iMaxTimer: in **milliseconds**. DTMF detection condition for timeout. 0 = no timeout

uStartByte: offset to start record audio in the sound file.

**Return:**

null

**Sample code:**

```
//Record a sound file
env.Send_RecordAudio(0, "c:\abc.wav", 0, "", 0, 0);
```

### 3.8.3 Send\_RecordAudio2

**Description:** Record **incoming and outgoing** voice into a sound file on the channel. This is two-way audio recording function, and it will mix in and out audio stream into one file. It supports DTMF detection conditions. Once the DTMF condition reached, the recording will stop, and On\_RecvAudioRecordDone event will be triggered to indicate which DTMF condition meets.

**Format:**

C++: void Send\_RecordAudio2(int ch, const char\* audioFileName, int iMaxDigit, const char\* termStr, int iMaxTimer, unsigned int uStartByte)  
 .NET: void Send\_RecordAudio2(int ch, string audioFileName, int iMaxDigit, string termStr, int iMaxTimer, unsigned int uStartByte)  
 OCX: void SendRecordAudio2(int ch, BSTR audioFileName, int iMaxDigit, BSTR termStr, int iMaxTimer, unsigned int uStartByte)  
 DLL: void GTAPI\_Send\_RecordAudio2(int ch, const char\* audioFileName, int iMaxDigit, const char\* termStr, int iMaxTimer, unsigned int uStartByte)

**Parameters:**

Ch: Channel Index

audioFileName: the sound file name

iMaxDigit: DTMF condition. 0=unlimited

termStr: DTMF detection condition.



iMaxTimer: in **milliseconds**. DTMF detection condition for timeout. 0 = no timeout  
 uStartByte: offset to start record audio in the sound file.

**Return:**

null

**Sample code:**

```
//Record audio into a sound file
env.Send_RecordAudio2(0, "c:\\abc.wav", 0, "", 0, 0);
```

### 3.8.4 Send\_AddAudio

**Description:** Sometimes you may need to play a list of files at one time. You can use this function to add as many as audio files you want, then use Send\_PlayAudio with "" in file name to play the list out.

**Format:**

C++: void Send\_AddAudio(int ch, const char\* pAudioName, unsigned int uBeginByte)  
 .NET: void Send\_AddAudio(int ch, const char\* pAudioName, unsigned int uBeginByte)  
 OCX: void SendAddAudio(int ch, const char\* pAudioName, unsigned int uBeginByte)  
 DLL: void GTAPI\_Send\_AddAudio

**Parameters:**

Ch: Channel Index  
 pAudioName: the sound file name  
 uStartByte: offset to start record audio in the sound file.

**Return:**

null

**Sample code:**

```
//play a list of sound files
env.Send_ClearAudio(0);
env.Send_AddAudio (0, "c:\\1.wav", 0);
env.Send_AddAudio (0, "c:\\2.wav", 0);
env.Send_AddAudio (0, "c:\\3.wav", 0);

env.Send_PlayAudio(0, "", 0, "", 0, 0);
```

### 3.8.5 Send\_ClearAudio

**Description:** Clear the audio list on the specific channel.

**Format:**

C++: void Send\_ClearAudio(int ch)  
.NET: void Send\_ClearAudio(int ch)  
OCX: void Send\_ClearAudio(int ch)  
DLL: void GTAPI\_Send\_ClearAudio

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//play a list of sound files
env.Send_ClearAudio(0);
env.Send_AddAudio (0, "c:\1.wav", 0);
env.Send_AddAudio (0, "c:\2.wav", 0);
env.Send_AddAudio (0, "c:\3.wav", 0);
env.Send_PlayAudio(0, "", 0, "", 0, 0);
```

### 3.8.6 Send\_StopAudio

**Description:** Stop the audio on the channel(Stop both playing and recording). *You **don't** need to explicitly use this method when call is disconnected. The SDK will stop audio recording or playing automatically once the channel is idle.*

**Format:**

C++: void Send\_StopAudio(int ch)  
.NET: void Send\_StopAudio(int ch)  
OCX: void Send\_StopAudio(int ch)  
DLL: void GTAPI\_Send\_StopAudio(int ch)

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```
//stop audio
env.Send_StopAudio(0);
```

### 3.8.7 Send\_StopAudioEx

**Description:** Stop the audio on the channel, either playing, recording, or both. *You don't need to explicitly use this method when call is disconnected. The SDK will stop audio recording or playing automatically once the channel is idle.*

If reason is specified, the reason will be sent back for developer to check. For example, developer may wish to know what kinds of reason were caused to call StopAudioEx, so he/she can do specific action according to the stop reason.

**Format:**

C++: void Send\_StopAudioEx(int ch, int opt, const char\* reason)

.NET: void Send\_StopAudioEx(int ch, int opt, string reason)

OCX: void Send\_StopAudioEx(int ch, int opt, BSTR reason)

DLL: void GTAPI\_Send\_StopAudioEx(int ch, int opt, const char\* reason)

**Parameters:**

**Ch:** Channel Index

**Opt:** 0 = stop both playing and recording. 1 = stop playing. 2 = stop recording

**Reason:** A reason to be sent back by On\_RecvAudioPlayDone and/or

On\_RecvAudioRecordDone events. When this function is called, for both above events, you will get the doneReason is GT\_AUDIO\_DONE\_FORCED\_STOP, with last parameter dtmfBuffer. In this case, dtmfBuffer is the same value of reason you called Send\_StopAudioEx.

**Return:**

null

**Sample code:**

```
//stop playing audio, with reason 'own timeout'.
env.Send_StopAudioEx(0, 1, "Own Timeout");
```

### 3.8.8 Send\_SetAudioFormat

**Description:** It is used to change the format of audio file when using Send\_RecordAudio or Send\_RecordAudio2.

## Format:

C++: void Send\_SetAudioFormat(int ch, GT\_UINT audioCode, GT\_UINT audioSample, GT\_UINT audioBit);  
.NET: void Send\_SetAudioFormat(int ch, uint audioCode, uint audioSample, uint audioBit);  
OCX: void Send\_SetAudioFormat(int ch, long audioCode, long audioSample, long audioBit);  
DLL: void GTAPI\_SetAudioFormat(int ch, GT\_UINT audioCode, GT\_UINT audioSampleRate, GT\_UINT audioBit);

## Parameters:

Ch: Channel Index

### AudioCode:

ADPCM	0x00000000
ADPCM_4_BIT	0x00000001
ADPCM_3_BIT	0x00000002
MU_LAW	0x00000003 /* Default */
A_LAW	0x00000004
PCM_16_BIT	0x00000005
PCM_8_BIT	0x00000006
PCM	0x00000007

### AudioSamplingRates:

4_KHz	0x00000001
6_KHz	0x00000002
8_KHz	0x00000003 /* Default */
11_KHz	0x00000004
22_KHz	0x00000005
44_KHz	0x00000006

### Bits:

BIT_2	0x00000010
BIT_3	0x00000020
BIT_4	0x00000030
BIT_5	0x00000040
BIT_6	0x00000050
BIT_8	0x00000060
BIT_16	0x00000070

## Return:

null

## Sample code:

```
//set all channels to record in PCM 8K16bit WAV
```

```

api.StartServer(); //After using StartServer function, you can set each channel's
audio format
for (int i = 0; i < max_channel_number - 1; i++)
    api.Send_SetAudioFormat(i, 7, 3, 112);

//set all channels to record in Mulaw 8K 8bit WAV
api.StartServer(); //After using StartServer function, you can set each channel's
audio format
for (int i = 0; i < max_channel_number - 1; i++)
    api.Send_SetAudioFormat(i, 3, 3, 96);

```

### 3.8.9 Send\_GetAudioStatus

**Description:** Retrive audio status on the channel. An event On\_Recv\_AudioStatus will be triggered after this functions is used.

**Format:**

```

C++: void Send_GetAudioStatus(int ch)
.NET: void Send_GetAudioStatus(int ch)
OCX: void SendGetAudioStatus(int ch)
DLL: void GTAPI_Send_GetAudioStatus(int ch)

```

**Parameters:**

Ch: Channel Index

**Return:**

null

**Sample code:**

```

//get audio status on channel 0
env.Send_GetAudioStatus(0);

```

### 3.8.10 Send\_StartDXAudio

**Description:** Start DX Audio Sound on the channel. This function is used by softphone application to connect the channel media and DirectX sound device, so user can talk with the far-end by computer sound card.

**Format:**

```

C++: void Send_StartDXAudio(int ch)
.NET: void Send_StartDXAudio(int ch)
OCX: void SendStartDXAudio(int ch)

```

DLL: void GTAPI\_Send\_StartDXAudio(int ch)

**Parameters:**

Ch: Channel Index. From 0 to max number of channel – 1.

**Return:**

null

**Sample code:**

```
//Start DirectX Sound once the call is connected
void On_RecvConnected(ch)
{
    env.Send_StartDXAudio(ch);
}
```

### 3.8.11 Send\_StopDXAudio

**Description:** Stop DX Audio Sound on the channel. **Note:** you don't need to explicitly use this function. Once the call is disconnected(by far-end, or by local using Send\_HangUp), the SDK will automatically stop DirectX sound on the channel.

**Format:**

C++: void Send\_StartDXAudio(int ch)  
.NET: void Send\_StartDXAudio(int ch)  
OCX: void SendStartDXAudio(int ch)  
DLL: void GTAPI\_Send\_StartDXAudio(int ch)

**Parameters:**

Ch: Channel Index. From 0 to max number of channel – 1.

**Return:**

null

**Sample code:**

```
//Stop DirectX Sound if you want to start music on hold on this channel
void On_MusicOnHold_Button_Click
{
    env.Send_StopDXAudio(ch);
    env.Send_StartMusicOnHold(ch, ....);
}

//Stop DirectX Sound and start to play a sound file to remote side
void On_PlaySound_Button_Click
```

```

{
    env.Send_StopDXAudio(ch);
    env.Send_PlayAudio(ch, .....);
}

```

### 3.8.12 Send\_ResetDXAudio

**Description:** Use this function to dynamically change sound device in a live call. This is a fantastical feature for softphone application. Sometimes the users of softphone(for example, call center agents) may need to use USB hearphone after the call is connected. This function provides the ability to switch the sound cards in a live call after DirectX is already opened.

Two tags are used to control sound devices.

"gtsrv.sip.dxsound.device.playback" //playback device keyword

"gtsrv.sip.dxsound.device.capture" //recording device keyword

You can even set different sound devices for playback and recording.

**Format:**

C++: void Send\_ResetDXAudio(int ch)

.NET: void Send\_ResetDXAudio(int ch)

OCX: void SendResetDXAudio(int ch)

DLL: void GTAPI\_Send\_ResetDXAudio(int ch)

**Parameters:**

Ch: Channel Index. From 0 to max number of channel – 1.

**Return:**

null

**Sample code:**

//Change sound devices and reset DirectX sound

CFG\_SetValue("gtsrv.sip.dxsound.device.playback", "2345M"); //set key word of playback device

CFG\_SetValue("gtsrv.sip.dxsound.device.capture", "SB879B"); //set key word of recording device

Send\_ResetDxAudio(ch);

### 3.8.13 Send\_SetDXAudioVolume

Not used any more.

### 3.8.14 On\_RecvAudioPlayDone

**Description:** Event to tell that playing sound(Send\_PlayAudio) is done.

**Format:**

C++: void On\_RecvAudioPlayDone(int ch, int doneReason, const char\* dtmfBuffer)

.NET: void On\_RecvAudioPlayDone(int ch, int doneReason, string dtmfBuffer)

OCX: void On\_RecvAudioPlayDone(int ch, int doneReason, BSTR dtmfBuffer)

DLL: void GTAPI\_SetCB\_On\_RecvAudioPlayDone

**Parameters:**

**Ch:** Channel Index(based on 0)

**doneReason:**

0 = GT\_AUDIO\_DONE\_DTMF\_TIMEOUT

1 = GT\_AUDIO\_DONE\_DTMF\_MAX\_DIGITS

2 = GT\_AUDIO\_DONE\_DTMF\_TERM\_DIGIT\_DETECTED

3 = GT\_AUDIO\_DONE\_PLAY

4 = GT\_AUDIO\_DONE\_RECORD //Only for On\_RecvAudioRecordDone

5 = GT\_AUDIO\_DONE\_FORCED\_STOP //used Send\_StopAudio or  
Send\_StopAudioEx or call is disconnected. If used Send\_StopAudioEx, the  
dtmfBuffer contains the reason called in Send\_StopAudioEx.

**dtmfBuffer:** DTMF string received during playing sound, or if called  
Send\_StopAudioEx, this is the reason set in Send\_StopAudioEx.

**Return:**

null

**Sample code:**

//Play another sound when last sound is done normally

```
void On_RecvAudioPlayDone(int ch, int doneReason, const char* dtmfBuffer)
{
    if(doneReason == 3)
    {
        Send_PlayAudio(ch, "c:\\abc.wav", 0, "", 0, 0);
    }
}
```

### 3.8.15 On\_RecvAudioRecordDone

**Description:** Event to tell that recording sound(Send\_RecordAudio) is done.

**Format:**

C++: void On\_RecvAudioRecordDone(int ch, int doneReason, const char\* dtmfBuffer)



.NET: void On\_RecvAudioRecordDone(int ch, int doneReason, string dtmfBuffer)  
 OCX: void On\_RecvAudioRecordDone(int ch, int doneReason, BSTR dtmfBuffer)  
 DLL: void GTAPI\_SetCB\_On\_RecvAudioRecordDone

**Parameters:**

**Ch:** Channel Index(based on 0)

**doneReason:**

0 = GT\_AUDIO\_DONE\_DTMF\_TIMEOUT  
 1 = GT\_AUDIO\_DONE\_DTMF\_MAX\_DIGITS  
 2 = GT\_AUDIO\_DONE\_DTMF\_TERM\_DIGIT\_DETECTED  
 3 = GT\_AUDIO\_DONE\_PLAY //Only for On\_RecvAudioPlayDone  
 4 = GT\_AUDIO\_DONE\_RECORD  
 5 = GT\_AUDIO\_DONE\_FORCED\_STOP //used Send\_StopAudio or  
 Send\_StopAudioEx or call is disconnected. If used Send\_StopAudioEx, the  
 dtmfBuffer contains the reason called in Send\_StopAudioEx.  
**dtmfBuffer:** DTMF string received during recording sound, or if called  
 Send\_StopAudioEx, this is the reason set in Send\_StopAudioEx.

**Return:**

null

**Sample code:**

```
//log out
void On_RecvAudioRecordDone(int ch, int doneReason, string dtmfBuffer)
{
    printf("recording is done");
}
```

### 3.8.16 On\_RecvAudioStatus

**Description:** Event to tell the status of audio. This event can be triggered by Send\_GetAudioStatus or when the status of audio is changed.

**Format:**

C++: void On\_RecvAudioStatus(int ch, int resType, int statusCode, unsigned long bytesDone)  
 .NET: void On\_RecvAudioStatus(int ch, int resType, int statusCode, unsigned long bytesDone)  
 OCX: void OnRecvAudioStatus(int ch, int resType, int statusCode, unsigned long bytesDone)  
 DLL: void GTAPI\_SetCB\_On\_RecvAudioStatus

**Parameters:**

**Ch:** Channel Index(based on 0)

resType:

0 = GT\_AUDIO\_RES\_BOTH

1 = GT\_AUDIO\_RES\_IN

2 = GT\_AUDIO\_RES\_OUT

statusCode:

0 = GT\_AUDIO\_STATUS\_IDLE

1 = GT\_AUDIO\_STATUS\_PLAYING

2 = GT\_AUDIO\_STATUS\_RECORDING

3 = GT\_AUDIO\_STATUS\_STOPPING

4 = GT\_AUDIO\_STATUS\_UNAVAILABLE

bytesDone: how many bytes it has recorded or played.

**Return:**

null

**Sample code:**

```
//log out
```

```
void On_RecvAudioRecordDone(int ch, int doneReason, string dtmfBuffer)
```

```
{
```

```
    printf("recording is done");
```

```
}
```

### 3.8.17 On\_RecvDXAudioStatus

**Description:** Event to indicate that the channel is recording. If you set tag "gtphone.audio.record.enabled" to 1 to record the DirectX channel, then you will get this event to indicate the recording starts, and filename of recording audio. After call is disconnected(you received IDLE event on this channel), you can use the file name to do some post process for recording file.

**Format:**

C++: void On\_RecvDXAudioStatus(int ch, int status, const char\* fn)

.NET: void On\_RecvDXAudioStatus(int ch, int status, string fn)

OCX: void OnRecvDXAudioStatus(int ch, int status, BSTR fn)

DLL: void GTAPI\_SetCB\_On\_RecvDXAudioStatus

**Parameters:**

Ch: Channel Index(based on 0)

status: 1 = recording

fn: file name of recording

**Return:**

null

**Sample code:**

```
//record the file name when received this event
void On_RecvDXAudioStatus (int ch, int status, const char* fn)
{
    string sFileName = fn;
}
//then when the call is done, copy the file to other place
void On_RecvIdle(int ch)
{
    CopyFile(sFileName, sDestFileName);
}
```

## 3.9 MOH(Music On Hold) Functions and Events

### 3.9.1 Send\_StartMusicOnHold

**Description:** This function starts music on the channel. Before you can use this function, you have to prepare a folder which only contains music wav files. (wav file format can be 8k 16bit pcm mono, or 8k 8bit mulaw/alaw mono)

**Format:**

C++: void Send\_StartMusicOnHold(int ch, const char\* sRoot, int bRandom, int maxTime)  
.NET: void Send\_StartMusicOnHold(int ch, string sRoot, int bRandom, int maxTime)  
OCX: void SendStartMusicOnHold(int ch, BSTR sRoot, int bRandom, int maxTime)  
DLL: void GTAPI\_Send\_StartMusicOnHold(int ch, const char\* sRoot, int bRandom, int maxTime)

**Parameters:**

Ch: Channel Index(based on 0)  
sRoot: folder of music files  
bRandom: 1 = random playing the music files. 0 = no, always same sequence  
maxTime: max time to play music on hold. 0 = unlimited. **Not implemented yet.**

**Return:**

null

**Sample code:**

```
//start music on hold
Send_StartMusicOnHold(0, "c:\moh", 1, 0);
```

### 3.9.2 Send\_StopMusicOnHold

**Description:** Stop music on channel

**Format:**

C++: void Send\_StopMusicOnHold(int ch)  
.NET: void Send\_StopMusicOnHold(int ch)  
OCX: void SendStopMusicOnHold(int ch)  
DLL: void GTAPI\_Send\_StopMusicOnHold(int ch)

**Parameters:**

Ch: Channel Index(based on 0)

**Return:**

null

**Sample code:**

```
//stop music on hold
Send_StopMusicOnHold(0);
```

### 3.9.3 Softphone “Hold” implementation

If your SIP server doesn't support “Music On Hold” feature for RE-INVITE message, you may be interested in implementing it on your softphone side. In order to implement music on hold, you have to setup a music file folder on the machine. There is only one wav file(somewhere1.wav) in SDK\audio folder. If you make yours, please make sure it has the same wav format as the one in SDK.

For example, assume c:\moh is the folder. Sample code is followed:

```
Bool bHolding = false;
```

```
On_ButtonHoldClick()
```

```
{
    if(!bHolding )
    {
        Send_StopDXAudio(currentChanIndex);
        Send_StartMusicOnHold(currentChanIndex, "c:\moh", 1, 0);
        bHolding = true;
    }
    else
    {
```

```

        Send_StopMusicOnHold(currentChanIndex);
        Send_StartDXAudio(currentChanIndex);
        bHolding = false;
    }
}

```

## 3.10 Conference Functions and Events

In order to use conference feature, you must set "gtsrv.sip.conference.room" to the number of conference room before env.StartServer.

### 3.10.1 Send\_StartConference

**Description:** Start a conference room. Only after it is started, you can add channels into this conference room.

**Format:**

```

C++: void Send_StartConference(int conf)
.NET: void Send_StartConference(int conf)
OCX: void SendStartConference(int conf)
DLL: void GTAPI_Send_StartConference(int conf)

```

**Parameters:**

conf: Conference Room Index(based on 0)

**Return:**

null

**Sample code:**

```

//start conference room 0
Send_StartConference(0);

```

### 3.10.2 Send\_StopConference

**Description:** Stop a conference room.

**Format:**

```

C++: void Send_StopConference(int conf)
.NET: void Send_StopConference(int conf)
OCX: void SendStopConference(int conf)
DLL: void GTAPI_Send_StopConference(int conf)

```

**Parameters:**

conf: Conference Room Index(based on 0)

**Return:**

null

**Sample code:**

```
//stop conference room 0
Send_ StopConference(0);
```

### 3.10.3 Send\_SetChanInConference

**Description:** Set channel into a conference room, or take it out from conference room.

**Format:**

```
C++: void Send_SetChanInConference(int conf, int ch, int bAdd)
.NET: void Send_SetChanInConference(int conf, int ch, int bAdd)
OCX: void Send_SetChanInConference(int conf, int ch, int bAdd)
DLL: void GTAPI_Send_SetChanInConference(int conf, int ch, int bAdd)
```

**Parameters:**

conf: Conference Room Index(based on 0)  
 ch: Channel Index(based on 0). *It can be -1, to indicate the local pc(sound device).*  
 bAdd: 1 = add the channel into conference. 0 = take the channel out from conference.

**Return:**

null

**Sample code:**

```
//set a channel into conference room
    if(Enable Conference)
    {
        //assume you only have two channels for the phone
        for(int i=0; i<2; i++)
        {
            if(GetChannel(i)->ch_status ==
GTAPI_Channel::GTAPI_CHANNEL_CONNECTED) //if this channel is connected
            {
                Send_StopDXAudio(i); //then stop dx audio on the
channel.

                Sleep(200);
                Send_SetChanInConference(0, i, 1); //put this
channel into conference room
```

```

        }
    }

    Send_SetChanInConference(0, -1, 1); //add dx audio(local pc) into
conference room

    }
    else //stop conference
    {
        Send_SetChanInConference(0, -1, 0); //take dx audio (local pc) out
of conference room

        for(int i=0; i<2; i++)
        {
            if(GetChannel(i)->ch_status ==
GTAPI_Channel::GTAPI_CHANNEL_CONNECTED)
            {
                Send_SetChanInConference(0, i, 0);
                Sleep(200);
                Send_StartDXAudio(i);
            }
        }
    }
}

```

*From V1.71f, you can dynamically create conference room when the application is running. Here are some functions for this purpose.*

### 3.10.4 CreateConf

**Description:** Create a conference room.

**Format:**

C++: unsigned long CreateConf();

.NET: unsigned long CreateConf()

OCX: long CreateConf()

DLL: unsigned long GTAPI\_CreateConf ()

**Parameters:**

null

**Return:**

Conference handle

**Sample code:**

```
unsigned long confHandle = CreateConf();
```

### 3.10.5 DestroyConf

**Description:** Destroy a conference room.

**Format:**

```
C++: void DestroyConf(unsigned long h);
.NET: void DestroyConf(ulong h);
OCX: void DestroyConf(long h);
DLL: void GTAPI_DestroyConf(unsigned long h);
```

**Parameters:**

h: The handle of conference

**Return:**

null

**Sample code:**

```
env.DestroyConf(confHandle);
```

### 3.10.6 SetChanInConf

**Description:** Destroy a conference room.

**Format:**

```
C++: BOOL SetChanInConf(unsigned long h, int ch, int bAdd);
.NET: BOOL SetChanInConf(ulong h, int ch, int bAdd);
OCX: BOOL SetChanInConf(long h, int ch, int bAdd);
DLL: BOOL GTAPI_SetChanInConf(unsigned long h, int ch, int bAdd);
```

**Parameters:**

h: The handle of conference

ch: The channel index. *It can be -1, to indicate local pc(sound device).*

bAdd: 1 = add                      The channel will be added into the conference

0 = out                          Set the channel out of the conference room

2 = monitor                      The channel will be able to monitor the conference room(means can hear the voice in conference, but cannot speak in conference room)

**Return:**

Boolean shows if the op is successful.



**Sample code:**

```
env.SetChanInConf(hConf, 0, 1); //add the channel 0 into the conference hConf.
```

### 3.10.7 GetConfIndex

**Description:** Get the index of conference handle.

**Format:**

```
C++: int GetConfIndex(unsigned long h);
.NET: int GetConfIndex(ulong h);
OCX: int GetConfIndex(long h);
DLL: int GTAPI_GetConfIndex(unsigned long h);
```

**Parameters:**

h: The handle of conference

**Return:**

The index of conference handle

**Sample code:**

```
int idx = env.GetConfIndex(hCOnf);
```

### 3.10.8 SetChanConfMask

**Description:** Set channel's output when in conference room

**Format:**

```
C++: void SetChanConfMask(int ch, unsigned int bitMask);
.NET: void SetChanConfMask(int ch, uint bitMask);
OCX: void SetChanConfMask(int ch, int bitMask);
DLL: void GTAPI_SetChanConfMask(int ch, unsigned int bitMask);
```

**Parameters:**

ch: channel index

bitMask: bit mask to enable or disable output

**Return:****Sample code:**

This function is used to disable the chan's output voice to other channels in the same conference.

Default channel mask is always 0xFFFFFFFF, which means output to all other channels in the conference room.

Every bit marks a channel. If the bit is 1, its voice can output to the channel.

The First channel in the conference room is 0x01.

The second channel in the conference room is 0x02.

The third channel in the conference room is 0x04.

So if you want the channel's output goes to the first channel, and the third channel, you can set this for this channel:

`SetChanConfMaskch, 0x05);` //which  $0x05 = 0x01 + 0x04$

Another example,

1st channel is connected with Agent. (Channel Index is 0, and it is the first channel set to the conference room)

2nd channel is connected with Customer. (Channel Index is 1, and it is the second channel set to the conference room)

3rd channel is supervisor. (Channel Index is 2, and it is the third channel set to the conference room)

They are all in the same conference room. Regularly if don't set anything, they can hear each other.

If supervisor only wants the agent hear his voice, not the customer, you can do so:

`SetChanConfMask(2, 0x01);`

It means that only the first channel get his voice.

## 3.11 Conference Audio Functions and Events

### 3.11.1 Send\_ConfPlayAudio

**Description:** Play a sound file on the conference.

**Format:**

C++: `void Send_ConfPlayAudio(int conf, const char* audioFileName, int iMaxDigit, const char* termStr, int iMaxTimer, unsigned int uStartByte)`

.NET: `void Send_ConfPlayAudio(int conf, string audioFileName, int iMaxDigit, string termStr, int iMaxTimer, unsigned int uStartByte)`

OCX: `void SendConfPlayAudioEx(int conf, BSTR audioFileName, int iMaxDigit, BSTR termStr, int iMaxTimer, unsigned int uStartByte);`

DLL: `void GTAPI_Send_ConfPlayAudio(int conf, const char* audioFileName, int iMaxDigit, const char* termStr, int iMaxTimer, unsigned int uStartByte)`

**Parameters:**

Conf: conference index, based on 0

audioFileName: the sound file name

iMaxDigit: DTMF condition. 0=unlimited  
termStr: DTMF detection condition.  
iMaxTimer: in **milliseconds**. DTMF detection condition for timeout. 0 = no timeout  
uStartByte: offset to start play audio in the sound file.

**Return:**

null

**Sample code:**

```
//Play a sound file in conference room 0
env.Send_ConfPlayAudio(0, "c:\\abc.wav", 0, "", 0, 0);
```

### 3.11.2 Send\_ConfRecordAudio

**Description:** Record conference voice into a sound file.

**Format:**

C++: void Send\_ConfRecordAudio(int conf, const char\* audioFileName, int iMaxDigit, const char\* termStr, int iMaxTimer, unsigned int uStartByte)  
.NET: void Send\_ConfRecordAudio(int conf, string audioFileName, int iMaxDigit, string termStr, int iMaxTimer, unsigned int uStartByte)  
OCX: void SendConfRecordAudioEx(int conf, BSTR audioFileName, int iMaxDigit, BSTR termStr, int iMaxTimer, unsigned int uStartByte)  
DLL: void GTAPI\_Send\_ConfRecordAudio(int conf, const char\* audioFileName, int iMaxDigit, const char\* termStr, int iMaxTimer, unsigned int uStartByte)

**Parameters:**

Conf: the index of conference  
audioFileName: the sound file name  
iMaxDigit: DTMF condition. 0=unlimited  
termStr: DTMF detection condition.  
iMaxTimer: in **milliseconds**. DTMF detection condition for timeout. 0 = no timeout  
uStartByte: offset to start record audio in the sound file.

**Return:**

null

**Sample code:**

```
//Record a sound file
env.Send_ConfRecordAudio(0, "c:\\abc.wav", 0, "", 0, 0);
```

### 3.11.3 Send\_ConfRecordAudio2

This function is the same as Send\_ConfRecordAudio.

### 3.11.4 Send\_ConfAddAudio

**Description:** Sometimes you may need to play a list of files at one time. You can use this function to add as many as audio files you want, then use Send\_ConfPlayAudio with "" in file name to play the list out.

**Format:**

C++: void Send\_ConfAddAudio(int conf, const char\* pAudioName, unsigned int uBeginByte)

.NET: void Send\_ConfAddAudio(int conf, string pAudioName, unsigned int uBeginByte)

OCX: void SendConfAddAudio(int conf, BSTR pAudioName, unsigned int uBeginByte)

DLL: void GTAPI\_Send\_ConfAddAudio

**Parameters:**

conf: the index of conference room

pAudioName: the sound file name

uStartByte: offset to start record audio in the sound file.

**Return:**

null

**Sample code:**

```
//play a list of sound files
env.Send_ConfClearAudio(0);
env.Send_ConfAddAudio (0, "c:\1.wav", 0);
env.Send_ConfAddAudio (0, "c:\2.wav", 0);
env.Send_ConfAddAudio (0, "c:\3.wav", 0);

env.Send_ConfPlayAudio(0, "", 0, "", 0, 0);
```

### 3.11.5 Send\_ConfClearAudio

**Description:** Clear the audio list on the specific conference.

**Format:**

C++: void Send\_ConfClearAudio(int conf)  
.NET: void Send\_ConfClearAudio(int conf)  
OCX: void Send\_ConfClearAudio(int conf)  
DLL: void GTAPI\_Send\_ConfClearAudio

**Parameters:**

Conf: the index of conference room

**Return:**

null

**Sample code:**

```
//play a list of sound files
env.Send_ConfClearAudio(0);
env.Send_ConfAddAudio (0, "c:\1.wav", 0);
env.Send_ConfAddAudio (0, "c:\2.wav", 0);
env.Send_ConfAddAudio (0, "c:\3.wav", 0);
env.Send_ConfPlayAudio(0, "", 0, "", 0, 0)
```

### 3.11.6 Send\_ConfStopAudio

**Description:** Stop the audio on the conference(Stop both playing and recording).

**Format:**

C++: void Send\_ConfStopAudio(int conf)  
.NET: void Send\_ConfStopAudio(int conf)  
OCX: void Send\_ConfStopAudio(int conf)  
DLL: void GTAPI\_Send\_ConfStopAudio(int conf)

**Parameters:**

conf: the index of conference

**Return:**

null

**Sample code:**

```
//stop audio
env.Send_ConfStopAudio(0);
```

### 3.11.7 Send\_ConfStopAudioEx

**Description:** Stop the audio on the conference, either stop playing, or stop recording, or both. If reason is specified, the reason will be sent back for developer to check. For example, developer may wish to know what kinds of reason were caused to call Send\_ConfStopAudioEx, so he/she can do specific action according to the stop reason.

**Format:**

C++: void Send\_ConfStopAudioEx(int conf, int opt, const char\* reason)  
.NET: void Send\_ConfStopAudioEx(int conf, int opt, string reason)  
OCX: void Send\_ConfStopAudioEx(int conf, int opt, BSTR reason)  
DLL: void GTAPI\_Send\_ConfStopAudioEx(int conf, int opt, const char\* reason)

**Parameters:**

**conf:** the index of conference

**opt:** 0 = stop both playing and recording. 1 = stop playing. 2 = stop recording.

**reason:** A reason to be sent back by On\_RecvConfAudioPlayDone and/or On\_RecvConfAudioRecordDone events. When this function is called, for both above events, you will get the doneReason is GT\_AUDIO\_DONE\_FORCED\_STOP, with last parameter dtmfBuffer. In this case, dtmfBuffer is the same value of reason you called Send\_StopAudioEx.

**Return:**

null

**Sample code:**

```
//stop playing audio, with reason "timeout"
env.Send_ConfStopAudioEx(0, 1, "timeout");
```

### 3.11.8 Send\_GetConfAudioStatus

**Description:** Retrive audio status on the conference. An event On\_Recv\_ConfAudioStatus will be triggered after this functions is used.

**Format:**

C++: void Send\_GetConfAudioStatus(int conf)  
.NET: void Send\_GetConfAudioStatus(int conf)  
OCX: void SendGetConfAudioStatus(int conf)  
DLL: void GTAPI\_Send\_GetConfAudioStatus(int conf)

**Parameters:**

conf: the index of conference

**Return:**

null

**Sample code:**

```
//get audio status on conference room 0
env.Send_GetConfAudioStatus(0);
```

### 3.11.9 Send\_ConfSetAudioFormat

**Description:** It is used to change the format of audio file when using Send\_ConfRecordAudio or Send\_ConfRecordAudio2.

**Format:**

C++: void Send\_ConfSetAudioFormat(int conf, GT\_UINT audioCode, GT\_UINT audioSample, GT\_UINT audioBit);  
.NET: void Send\_ConfSetAudioFormat(int conf, uint audioCode, uint audioSample, uint audioBit);  
OCX: void Send\_ConfSetAudioFormat(int conf, long audioCode, long audioSample, long audioBit);  
DLL: void GTAPI\_ConfSetAudioFormat(int conf, GT\_UINT audioCode, GT\_UINT audioSampleRate, GT\_UINT audioBit);

**Parameters:**

Conf: Conference Index

**AudioCode:**

ADPCM	0x00000000
ADPCM_4_BIT	0x00000001
ADPCM_3_BIT	0x00000002
MU_LAW	0x00000003 /* Default */
A_LAW	0x00000004
PCM_16_BIT	0x00000005
PCM_8_BIT	0x00000006
PCM	0x00000007

**AudioSamplingRates:**

4_KHz	0x00000001
6_KHz	0x00000002
8_KHz	0x00000003 /* Default */
11_KHz	0x00000004
22_KHz	0x00000005
44_KHz	0x00000006

Bits:	
BIT_2	0x00000010
BIT_3	0x00000020
BIT_4	0x00000030
BIT_5	0x00000040
BIT_6	0x00000050
BIT_8	0x00000060
BIT_16	0x00000070

**Return:**

null

**Sample code:**

```
//set to record in PCM 8K16bit WAV
api.Send_ConfSetAudioFormat(0, 7, 3, 112);
```

```
//set to record in Mulaw 8K 8bit WAV
api.Send_SetAudioFormat(0, 3, 3, 96);
```

### 3.11.10 On\_RecvConfAudioStatus

**Description:** Event to tell audio status of the conference. This event can be triggered by Send\_GetConfAudioStatus or when the status of audio is changed.

**Format:**

C++: void On\_RecvConfAudioStatus(int conf, int resType, int statusCode, unsigned long bytesDone)

.NET: void On\_RecvConfAudioStatus(int conf, int resType, int statusCode, unsigned long bytesDone)

OCX: void OnRecvConfAudioStatus(int conf, int resType, int statusCode, unsigned long bytesDone)

DLL: void GTAPI\_SetCB\_On\_RecvConfAudioStatus

**Parameters:**

conf: the index of conference room

resType:

0 = GT\_AUDIO\_RES\_BOTH

1 = GT\_AUDIO\_RES\_IN

2 = GT\_AUDIO\_RES\_OUT

statusCode:

0 = GT\_AUDIO\_STATUS\_IDLE

1 = GT\_AUDIO\_STATUS\_PLAYING



2 = GT\_AUDIO\_STATUS\_RECORDING  
 3 = GT\_AUDIO\_STATUS\_STOPPING  
 4 = GT\_AUDIO\_STATUS\_UNAVAILABLE  
 bytesDone: how many bytes it has recorded or played.

**Return:**

null

**Sample code:**

```
//log out
void On_RecvConfAudioStatus (int conf, int resType, int statusCode, unsigned long
bytesDone)
{
    printf("got event");
}
```

### 3.11.11 On\_RecvConfAudioPlayDone

**Description:** Event to tell that playing sound(Send\_PlayAudio) is done.

**Format:**

C++: void On\_RecvConfAudioPlayDone(int conf, int doneReason, const char\* dtmfBuffer)

.NET: void On\_RecvConfAudioPlayDone(int conf, int doneReason, string dtmfBuffer)

OCX: void On\_RecvConfAudioPlayDone(int conf, int doneReason, BSTR dtmfBuffer)

DLL: void GTAPI\_SetCB\_On\_RecvConfAudioPlayDone

**Parameters:**

**conf:** conference index

**doneReason:**

0 = GT\_AUDIO\_DONE\_DTMF\_TIMEOUT

1 = GT\_AUDIO\_DONE\_DTMF\_MAX\_DIGITS

2 = GT\_AUDIO\_DONE\_DTMF\_TERM\_DIGIT\_DETECTED

3 = GT\_AUDIO\_DONE\_PLAY

4 = GT\_AUDIO\_DONE\_RECORD //Only for On\_RecvAudioRecordDone

5 = GT\_AUDIO\_DONE\_FORCED\_STOP //used Send\_StopAudio or call is disconnected. If used Send\_ConfStopAudioEx, the dtmfBuffer contains the reason called in Send\_StopAudioEx.

**dtmfBuffer:** DTMF string received during playing sound, or if called Send\_ConfStopAudioEx, this is the reason set in Send\_ConfStopAudioEx.

**Return:**

null

**Sample code:**

```
//Play another sound when last sound is done normally
void On_RecvConfAudioPlayDone(int conf, int doneReason, const char*
dtmfBuffer)
{
    if(doneReason == 3)
    {
        Send_PlayAudio(conf, "c:\\abc.wav", 0, "", 0, 0);
    }
}
```

**3.11.12 On\_RecvConfAudioRecordDone**

**Description:** Event to tell that recording sound(Send\_RecordAudio) is done.

**Format:**

C++: void On\_RecvConfAudioRecordDone(int conf, int doneReason, const char\* dtmfBuffer)

.NET: void On\_RecvConfAudioRecordDone(int conf, int doneReason, string dtmfBuffer)

OCX: void On\_RecvConfAudioRecordDone(int conf, int doneReason, BSTR dtmfBuffer)

DLL: void GTAPI\_SetCB\_On\_RecvConfAudioRecordDone

**Parameters:**

**conf:** the index of conference

**doneReason:**

0 = GT\_AUDIO\_DONE\_DTMF\_TIMEOUT

1 = GT\_AUDIO\_DONE\_DTMF\_MAX\_DIGITS

2 = GT\_AUDIO\_DONE\_DTMF\_TERM\_DIGIT\_DETECTED

3 = GT\_AUDIO\_DONE\_PLAY //Only for On\_RecvAudioPlayDone

4 = GT\_AUDIO\_DONE\_RECORD

5 = GT\_AUDIO\_DONE\_FORCED\_STOP //used Send\_StopAudio or call is disconnected.. If used Send\_ConfStopAudioEx, the dtmfBuffer contains the reason called in Send\_StopAudioEx.

**dtmfBuffer:** DTMF string received during recording sound, or if called Send\_ConfStopAudioEx, this is the reason set in Send\_ConfStopAudioEx.

**Return:**

null

**Sample code:**

```
//log out
void On_RecvConfAudioRecordDone(int conf, int doneReason, string dtmfBuffer)
{
    printf("recording is done");
}
```

## 3.12 Tone Detection Functions and Events

### 3.12.1 Send\_AddTone

**Description:** Add a tone to detect.

**Format:**

```
C++: void Send_AddTone(int ch, int freq, int duration);
.NET: void Send_AddTone(int ch, int freq, int duration);
OCX: void Send_AddTone(int ch, int freq, int duration);
DLL: void GTAPI_Send_AddTone(int ch, int freq, int duration);
```

**Parameters:**

ch: Channel Index(based on 0)  
 freq: Frequency to detect, in HZ.  
 duration: the period of the tone, in milliseconds

**Return:**

null

**Sample code:**

```
//to detect fax tone
env.Send_AddTone(0, 1100, 400); //1100HZ for 400ms. T30 standard for caller
env.Send_AddTone(0, 2100, 2400); //2100HZ for 2400ms. T30 standard for callee
```

### 3.12.2 Send\_ClearToneList

**Description:** Clear the tone list for the channel

**Format:**

```
C++: void Send_ClearToneList(int ch);
.NET: void Send_ClearToneList(int ch);
OCX: void Send_ClearToneList(int ch);
DLL: void Send_ClearToneList(int ch);
```

**Parameters:**

ch: Channel Index(based on 0)

**Return:**

null

### 3.12.3 Send\_StartToneDetection

**Description:** Start to detect tones on the channel

**Format:**

C++: void Send\_StartToneDetection(int ch);  
.NET: void Send\_StartToneDetection(int ch);  
OCX: void Send\_StartToneDetection(int ch);  
DLL: void GTAPI\_Send\_StartToneDetection(int ch);

**Parameters:**

ch: Channel Index(based on 0)

**Return:**

null

**Sample code:**

```
//to detect fax tone  
env.Send_AddTone(0, 1100, 400); //1100HZ for 400ms. T30 standard for caller  
env.Send_AddTone(0, 2100, 2400); //2100HZ for 2400ms. T30 standard for callee  
env.Send_StartToneDetection(0);
```

### 3.12.4 Send\_StopToneDetection

**Description:** Stop detecting tones on the channel

**Format:**

C++: void Send\_StopToneDetection(int ch);  
.NET: void Send\_StopToneDetection(int ch);  
OCX: void Send\_StopToneDetection(int ch);  
DLL: void GTAPI\_Send\_StopToneDetection(int ch);

**Parameters:**

ch: Channel Index(based on 0)

**Return:**

null

**Sample code:**

```
env.Send_StopToneDetection(0);
```

### 3.12.5 On\_RecvToneDetected

**Description:** Event to indicate that the tone is detected

**Format:**

```
C++: void On_RecvToneDetected(int ch, int freq);
.NET: void On_RecvToneDetected(int ch, int freq);
OCX: void On_RecvToneDetected(int ch, int freq);
DLL: void GTAPI_SetCB_On_RecvToneDetected
```

**Parameters:**

ch: Channel Index(based on 0)  
freq: The frequency that is detected.

**Return:**

null

## 3.13 VAD Functions and Events

Please set "gtsrv.sip.on.in.vad" to 1 if you want to detect incoming voice, and set "gtsrv.sip.on.out.vad" to 1 if you want to detect outgoing voice. These two tags are static, and have to be set before StartServer. If you want to dynamically enable the VAD on specific channel, you can use Send\_EnableVAD and Send\_DisableVAD.

### 3.13.1 On\_VoiceActivityDetected

**Description:** Event to notify the Voice Activity.

**Format:**

```
C++: void On_VoiceActivityDetected(int ch, int voice_dir, int voice_on, int level,
unsigned int reserved)
.NET: void On_VoiceActivityDetected(int ch, int voice_dir, int voice_on, int level,
unsigned int reserved)
```

OCX: void OnVoiceActivityDetected(int ch, int voice\_dir, int voice\_on, int level, unsigned int reserved)  
 DLL: void GTAPI\_SetCB\_On\_VoiceActivityDetected

**Parameters:**

ch: Channel Index(based on 0)  
 voice\_dir: 1 = incoming, 0 = outgoing  
 voice\_on: 1 = voice, 0 = silence  
 level: voice energy level. Always 0 now.  
 reserved: not used.

**Return:**

null

**Sample code:**

```
//log out event
void On_VoiceActivityDetected(int ch, int voice_dir, int voice_on, int level, unsigned
int reserved)
{
    if(voice_dir == 1)
    {
        if(voice_on == 1)
        {
            Printf("voice is on");
        }
        Else
        {
            Printf("voice is off");
        }
    }
}
```

### 3.13.2 Send\_EnableVAD

**Description:** Enable VAD on the channel

**Format:**

C++: void Send\_EnableVAD(int ch);  
 .NET: void Send\_EnableVAD(int ch);  
 OCX: void Send\_EnableVAD(int ch);  
 DLL: void GTAPI\_Send\_EnableVAD(int ch);

**Parameters:**

ch: Channel Index(based on 0)

**Return:**

null

### 3.13.3 Send\_DisableVAD

**Description:** Disable VAD on the channel

**Format:**

C++: void Send\_DisableVAD(int ch);  
.NET: void Send\_DisableVAD(int ch);  
OCX: void Send\_DisableVAD(int ch);  
DLL: void GTAPI\_Send\_DisableVAD(int ch);

**Parameters:**

ch: Channel Index(based on 0)

**Return:**

null

## 3.14 MWI(Message Waiting Indicator)

SDK supports SIP NOTIFY message from SIP PBX, to indicate softphone if there is voice mail.

### 3.14.1 On\_RecvNotifySimpleMsgSummary

**Description:** This event is triggered when a notify summary message is arrived.

**Format:**

C++: void On\_RecvNotifySimpleMsgSummary(const char\* sMsgWaiting, const char\* sMsgAccount, const char\* sVoiceMsg)  
.NET: void On\_RecvNotifySimpleMsgSummary(string sMsgWaiting, string sMsgAccount, string sVoiceMsg)  
OCX: void OnRecvNotifySimpleMsgSummary(BSTR sMsgWaiting, BSTR sMsgAccount, BSTR sVoiceMsg)

DLL: void OnRecvNotifySimpleMsgSummary(const char\* sMsgWaiting, const char\* sMsgAccount, const char\* sVoiceMsg)

**Parameters:**

sMsgWaiting: how many messages are waiting

sMsgAccount: the account number to dial for retriving the voice mail

sVoiceMsg: voice message info

**Return:**

null

## 3.15 Instant Message Fonctions and Events

SDK supports SIP MESSAGE, which is used to send and receive short instant message.

### 3.15.1 Send\_MessageText

**Description:** Send an instant message.

**Format:**

C++: void Send\_MessageText(int msg\_id, const char\* msgfrom, const char\* msgto, const char\* content)

.NET: void Send\_MessageText(int msg\_id, string msgfrom, string msgto, string content)

OCX: void SendMessageText(int msg\_id, BSTR msgfrom, BSTR msgto, BSTR content)

DLL: void GTAPI\_Send\_MessageText(int msg\_id, BSTR msgfrom, BSTR msgto, BSTR content)

**Parameters:**

msg\_id: the unique id that distinguishes the messages. This id will be returned later by On\_RecvMessageTextDelivered event to indicate if the message was successfully sent.

msgfrom: message from sip address. format is : <sip:abc@def.com>

msgto: message to sip address. format is : <sip:456@def.com>

content: instant message. for example: "Watson, come here."

**Return:**

null

**Sample code:**



```
Send_MessageText(1, "<sip:Bob@pcbest.net>", "<sip:Watson@pcbest.net>",
"Watson, come here.")
```

### 3.15.2 On\_RecvMessageTextDelivered

**Description:** Event to tell if the previous Send\_MessageText succeeded.

**Format:**

```
C++: void On_RecvMessageTextDelivered(int msg_id, int msg_code, const char*
msg_txt)
.NET: void On_RecvMessageTextDelivered(int msg_id, int msg_code, string
msg_txt)
OCX: void OnRecvMessageTextDelivered(int msg_id, int msg_code, BSTR msg_txt)
DLL: void GTAPI_SetCB_On_RecvMessageTextDelivered
```

**Parameters:**

msg\_id: the unique id that distinguishes the messages. This id was the one in Send\_MessageText  
msg\_code: 200 is OK(Successful). others are not successful.  
msg\_txt: message description.

**Return:**

null

**Sample code:**

### 3.15.3 On\_RecvMessageText

**Description:** Event to tell that a new instant message arrived.

**Format:**

```
C++: void On_RecvMessageText(int ch, const char* sFrom, const char* sTo, const
char* sDestAddr, const char* sViaAddr, const char* sContent)
.NET: void On_RecvMessageText(int ch, string sFrom, string To, string sDestAddr,
string sViaAddr, string sContent)
OCX: void OnRecvMessageText(int ch, BSTR sFrom, BSTR sTo, BSTR sDestAddr,
BSTR sViaAddr, BSTR sContent)
DLL: void GTAPI_SetCB_On_RecvMessageText
```

**Parameters:**

ch: channel index, from 0. if it is -1. the message is not attached to any exist call sessions.  
sFrom: message from

sTo: message to  
sDestAddr: message is for  
sViaAddr: message via  
sContent: message content

**Return:**

null

**Sample code:**

## 3.16 Support for SIP/Presence

Use CFG\_SetValue to turn on SIP "SUBSCRIBE" message for presence:

```
"gtsrv.sip.reg1.subscribe" = "1" //default it is 0(off)
"gtsrv.sip.reg1.subscribe.addr" = "123@sipproxy.com" //You can specify multiple
address. like "123@sipproxy.com;456@sipproxy.com;789@sipproxy.com"
"gtsrv.sip.reg1.subscribe.expire" = 600 //default it is 600, 10 minutes.
"gtsrv.sip.reg1.subscribe.accept" = 0 //0 = pdf(default), 1 = dialog-info
```

There are two events for subscribe and notify presence.

### 3.16.1 RecvSubscribeStatus

**Description:** Event to tell if subscribe to SIP server was successful.

**Format:**

```
C++: void On_RecvSubscribeStatus(int id, int status, int regtime)
.NET: void On_RecvSubscribeStatus(int id, int status, int regtime)
OCX: void OnRecvSubscribeStatus(int id, int status, int regtime)
DLL: void GTAPI_SetCB_On_RecvSubscribeStatus
```

**Parameters:**

id : sip account profile id, from 0  
status: 1 = successfully subscribe, 0 = failed  
regtime: seconds to subscribe

**Return:**

null

**Sample code:**

### 3.16.2 RecvNotifyPresence

**Description:** Event to tell if subscribe to SIP server was successful.

**Format:**

C++: void On\_RecvNotifyPresence(int id, const char\* subscription\_state, const char\* content\_type, const char\* content\_info)

.NET: void On\_RecvNotifyPresence(int id, string subscription\_state, string content\_type, string content\_info)

OCX: void On\_RecvNotifyPresence(int id, BSTR subscription\_state, BSTR content\_type, BSTR content\_info)

DLL: void GTAPI\_SetCB\_On\_RecvNotifyPresence

**Parameters:**

id : sip account profile id, from 0

subscription\_state: from SIP Notify message, it is "active" usually.

content\_type: from SIP Notify message, can be "application/dialog-info+xml", or "application/pdf+xml"

content\_info: real xml content

**Return:**

null

**Sample code:**

### 3.17 RTP and DirecX Audio Data Events

Set "gtsrv.sip.on.rtp.packet" to 1, or 2, or 3 to access rtp data. Default 0 is disabled.

1 = MULAW

2 = ALAW

3 = PCM

Set "gtsrv.sip.on.dx.audio" to 1, or 2, or 3 to access directx audio data. Default 0 is disabled.

1 = MULAW

2 = ALAW

3 = PCM

**Important Note:** If you enable this feature, you will get these events triggered every 20 milliseconds. Because the thread to trigger those functions is low level thread of span,

you should NOT do a lot of things in these events. For example, no loop, no sleep, no block functions like write buffer into disk, or no any operations may take too long.

**RTP audio events are also the interfaces to set your own RTP audio streams.** For example, for ASR(Auto Speech Recognition) application, you may need to access in real-time in memory instead of recording it into disk and sending the audio file to ASR engine. The same for TTS, you may need to change the outgoing RTP stream to play your TTS audio. The following two RTP audio streams let you **access and change** the RTP audio stream.

### 3.17.1 On\_RecvRTPPacket

**Description:** Event to indicate that received incoming RTP packet. **You can use this event to handle incoming audio stream in real-time to implement some special features like, ASR(Auto Speech Recognition), ISDN gateway, or special audio hardware.**

**Format:**

C++: void On\_RecvRTPPacket(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp, SYSTEMTIME\* pSysTime)

.NET: void On\_RecvRTPPacket(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

OCX: void OnRecvRTPPacket(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

DLL: void GTAPI\_SetCB\_On\_RecvRTPPacket

**Parameters:**

ch: Channel Index(based on 0)

fmt: This is what you set to access audio format

1 = MULAW

2 = ALAW

3 = PCM

buf: buffer pointer. The buffer is changeable.

buflen: buffer length

seq: sequence number

timestamp: timestamp

pSysTime: system time in millisecond

**Return:**

null

**Sample code:**

```
//copying buffer to somewhere else
```

```

void On_RecvRTPPacket(int ch, int fmt, char* buf, int buflen, unsigned short seq,
unsigned int timestamp, SYSTEMTIME* pSysTime)
{
    //assume buf1 is the memory buffer to hold incoming voice stream
    //later on, other thread will take the buffer out from buf1, and do your special
audio process
    memcpy(buf1, buf, buflen); //C++ code, memory copy
    buf1 += buflen; //move buf1 ahead of buflen bytes.
}

```

### 3.17.2 On\_SentRTPPacket

**Description:** Event to indicate sent a RTP packet out. **You can use this event to handle outgoing audio stream in real-time to implement some special features like, TTS(Text To Speech), ISDN gateway, or special audio hardware. The audio buffer passed by the event can be changed by using your audio data.**

**Format:**

C++: void On\_SentRTPPacket(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp, SYSTEMTIME\* pSysTime)

.NET: void On\_SentRTPPacket(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

OCX: void OnSentRTPPacket(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

DLL: void GTAPI\_SetCB\_On\_SentRTPPacket

**Parameters:**

ch: Channel Index(based on 0)

fmt: This is what you set to access audio format

1 = MULAW

2 = ALAW

3 = PCM

buf: buffer pointer. The buffer is changeable.

buflen: buffer length

seq: sequence number

timestamp: timestamp

pSysTime: system time in millisecond

**Return:**

null

**Sample code:**

//Insert your own outbound audio

```

void On_SentRTPPacket(int ch, int fmt, char* buf, int buflen, unsigned short seq,
unsigned int timestamp, SYSTEMTIME* pSysTime)
{
    //assume that buf1 is holding your outbound audio stream
    memcpy(buf, buf1, buflen);    //copy buflen bytes into buf
    buf1 += buflen; //buf1's pointer address move ahead buflen bytes for next access
}

```

### 3.17.3 On\_CaptureDXAudio

**Description:** Event to indicate captured DirectX audio. You can change the audio buffer to implement your special audio application.

**Format:**

C++: void On\_CaptureDXAudio(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp, SYSTEMTIME\* pSysTime)

.NET: void On\_CaptureDXAudio(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

OCX: void OnCaptureDXAudio(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

DLL: void GTAPI\_SetCB\_On\_CaptureDXAudio

**Parameters:**

ch: Channel Index(based on 0)

fmt: This is what you set to access audio format

1 = MULAW

2 = ALAW

3 = PCM

buf: buffer pointer

buflen: buffer length

seq: sequence number

timestamp: timestamp

pSysTime: system time in millisecond

**Return:**

null

**Sample code:**

//copying buffer to somewhere else

```

void On_CaptureDXAudio(int ch, int fmt, char* buf, int buflen, unsigned short seq,
unsigned int timestamp, SYSTEMTIME* pSysTime)
{
    memcpy(buf1, buf, buflen);
}

```

### 3.17.4 On\_RenderDXAudio

**Description:** Event to indicate played DirectX audio. You can change the audio buffer to implement your special audio application.

**Format:**

C++: void On\_RenderDXAudio(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp, SYSTEMTIME\* pSysTime)

.NET: void On\_RenderDXAudio(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

OCX: void OnRenderDXAudio(int ch, int fmt, char\* buf, int buflen, unsigned short seq, unsigned int timestamp)

DLL: void GTAPI\_SetCB\_On\_RenderDXAudio

**Parameters:**

ch: Channel Index(based on 0)

fmt: This is what you set to access audio format

1 = MULAW

2 = ALAW

3 = PCM

buf: buffer pointer

buflen: buffer length

seq: sequence number

timestamp: timestamp

pSysTime: system time in millisecond

**Return:**

null

**Sample code:**

```
//copying buffer to somewhere else
void On_RenderDXAudio(int ch, int fmt, char* buf, int buflen, unsigned short seq,
unsigned int timestamp, SYSTEMTIME* pSysTime)
{
    memcpy(buf1, buf, buflen);
}
```

### 3.17.5 On\_RTPRawPacket

Set "gtsrv.sip.on.rtp.packet" to -1 to trigger this event.

**Description:** This event can be used to get the RAW RTP packets received.

**Format:**

C++: void On\_RTPTRawPacket(int ch, int dir, unsigned char\* rawData, int dataLen, const char\* senderAddr, unsigned short senderPort, unsigned int tick)  
.NET: void On\_RTPTRawPacket(int ch, int dir, IntPtr rawData, int dataLen, string senderAddr, ushort senderPort, uint tick)  
OCX: void On\_RTPTRawPacket(int ch, int dir, IntPtr rawData, int dataLen, BSTR senderAddr, ushort senderPort, uint tick)  
DLL: void GTAPI\_SetCB\_On\_RTPTRawPacket

**Parameters:**

ch: Channel Index(based on 0)  
dir: 0 = received 1 = sent  
rawData: RTP raw buffer pointer.  
dataLen: buffer length  
senderAddr: sender's ip address  
senderPort: sender's ip port  
tick: system tick in millisecond

**Return:**

null

**Sample code:**

```
//C++, copying buffer to somewhere else
void On_RTPTRawPacket(int ch, int dir, unsigned char* rawData, int dataLen, const
char* senderAddr, unsigned short senderPort, unsigned int tick)
{
    //assume buf1 is the memory buffer
    memcpy(buf1, rawData, dataLen); //C++ code, memory copy
}

//C#
void On_RTPTRawPacket(int ch, int dir, IntPtr rawData, int dataLen, string
senderAddr, ushort senderPort, uint tick)
{
    unsafe{
        byte* p = (byte*)buf;
        for(int i=0; i<buflen; i++)
            val = *p;
    }
}

//VB.NET
void On_RTPTRawPacket(int ch, int dir, IntPtr rawData, int dataLen, string
senderAddr, ushort senderPort, uint tick)
```



```

{
    VB.NET code
    dim val(buflen) as Byte
    Marshal.Copy(buf, val, 0, buflen)
    For i = 0 To size - 1 Step 1
        pixVal = val(i)
    Next
}

```

## 3.18 Softphone Assistant Functions

### 3.18.1 PlayNumTone

**Description:** Play number tone in local sound card. When users click on the number buttons on softphone, the softphone may need to play a number tone sound in sound device to make it real. This function is used to play number tone 0-9, \* and #. **Note:** if you want to send DTMF key to remote side, you should use Send\_PlayDTMFStr function.

**Format:**

C++: bool PlayNumTone(char num)  
.NET: bool PlayNumTone(char num)  
OCX: bool PlayNumTone(char num)  
DLL: bool GTAPI\_PlayNumTone(char num)

**Parameters:**

num: '0'-'9', '\*', '#'

**Return:**

True if succeed, Otherwise false.

**Sample code:**

```

//Play number tone '0' in local computer
Env.PlayNumTone('0');
```

### 3.18.2 PlayLocalRingSound

**Description:** Play ring sound in local computer sound card. It will keep playing the ring sound until StopSound function is invoked.

**Format:**

C++: bool PlayLocalRingSound()  
.NET: bool PlayLocalRingSound()  
OCX: bool PlayLocalRingSound()  
DLL: bool GTAPI\_PlayLocalRingSound()

**Parameters:****Return:**

True if succeed, Otherwise false.

**Sample code:**

```
//play ring sound when there is an new incoming call
void On_RecvOffered(int ch, ..... )
{
    env.PlayLocalRingSound();
}
```

### 3.18.3 PlayRemoteRingSound

**Description:** Play ring sound in local computer to indicate that remote side is ringing. It will keep playing the ring sound until StopSound function is invoked.

**Format:**

C++: bool PlayRemoteRingSound ()  
.NET: bool PlayRemoteRingSound ()  
OCX: bool PlayRemoteRingSound ()  
DLL: bool GTAPI\_PlayRemoteRingSound()

**Parameters:****Return:**

True if succeed, Otherwise false.

**Sample code:**

```
//play ring sound when remote side start ringing
void On_RecvRinging(int ch)
{
    env.PlayRemoteRingSound();
}
```

### 3.18.4 PlayBusySound

**Description:** Play busy sound in local computer. It will keep playing the ring sound until StopSound function is invoked.

**Format:**

C++: bool PlayBusySound()  
.NET: bool PlayBusySound()  
OCX: bool PlayBusySound()  
DLL: bool GTAPI\_PlayBusySound()

**Parameters:**

**Return:**

True if succeed, Otherwise false.

**Sample code:**

```
//play busy sound when channel is idle
void On_RecvIdle(int ch)
{
    env.PlayBusySound();
}
```

### 3.18.5 StopSound

**Description:** Stop sound.

**Format:**

C++: bool StopSound()  
.NET: bool StopSound()  
OCX: bool StopSound()  
DLL: bool GTAPI\_StopSound()

**Parameters:**

**Return:**

True if succeed, Otherwise false.

**Sample code:**

```
//stop playing any sound when call is connected
void On_RecvConnected(int ch)
{
    env.StopSound();
}
```

```
}
```

### 3.18.6 SetMicVolume

**Description:** Set microphone's volume. *Note: You must use SetMainWnd to set the Windows handle. Please see the description of SetMainWnd.*

**Format:**

```
C++: void SetMicVolume(float v)
.NET: void SetMicVolume(float v)
OCX: void SetMicVolume(float v)
DLL: void GTAPI_SetMicVolume(float v)
```

**Parameters:**

v: volume. It is 0.00 – 1.00.

**Return:**

null

**Sample code:**

```
//Set microphone to max
env.SetMicVolume(1.00);
```

### 3.18.7 GetMicVolume

**Description:** Get microphone's volume. *Note: You must use SetMainWnd to set the Windows handle. Please see the description of SetMainWnd.*

**Format:**

```
C++: float GetMicVolume()
.NET: float GetMicVolume()
OCX: float GetMicVolume()
DLL: float GetMicVolume()
```

**Parameters:**

**Return:**

0.00 – 1.00

**Sample code:**

```
//Get microphone's volume
env.GetMicVolume();
```

### 3.18.8 SetSpeakerVolume

**Description:** Set speaker's volume. *Note: You must use SetMainWnd to set the Windows handle. Please see the description of SetMainWnd.*

**Format:**

C++: void SetSpeakerVolume(float v)  
.NET: void SetSpeakerVolume(float v)  
OCX: void SetSpeakerVolume(float v)  
DLL: void GTAPI\_SetSpeakerVolume(float v)

**Parameters:**

v: volume. It is 0.00 – 1.00.

**Return:**

null

**Sample code:**

```
//Set speaker to max
env.SetSpeakerVolume(1.00);
```

### 3.18.9 GetSpeakerVolume

**Description:** Get speaker's volume. *Note: You must use SetMainWnd to set the Windows handle. Please see the description of SetMainWnd.*

**Format:**

C++: float GetSpeakerVolume()  
.NET: float GetSpeakerVolume()  
OCX: float GetSpeakerVolume()  
DLL: float GetSpeakerVolume()

**Parameters:**

**Return:**

0.00 – 1.00

**Sample code:**

```
//Get speaker's volume
env.GetSpeakerVolume();
```

### 3.18.10 SetSpeakerMuteStatus

**Description:** Set if mute the speaker

**Format:**

C++: void SetSpeakerMuteStatus(BOOL bMute)  
.NET: void SetSpeakerMuteStatus(uint bMute)  
OCX: void SetSpeakerMuteStatus(long bMute)  
DLL: void GTAPI\_SetSpeakerMuteStatus(BOOL bMute)

**Parameters:**

bMute: 1 = Mute the speaker    0 = unmute the speaker

**Return:**

void

**Sample code:**

```
//Mute the speaker
env. SetSpeakerMuteStatus(1);
```

### 3.18.11 GetSpeakerMuteStatus

**Description:** Get the speaker's mute status.

**Format:**

C++: BOOL GetSpeakerMuteStatus();  
.NET: uint GetSpeakerMuteStatus();  
OCX: long GetSpeakerMuteStatus();  
DLL: BOOL GTAPI\_GetSpeakerMuteStatus();

**Parameters:**

null

**Return:**

1 = speaker is muted  
0 = speaker is in normal status

**Sample code:**

```
if(env.GetSpeakerMuteStatus() == 1) { // Speaker is in mute status }
```

### 3.18.12 SetMicMuteStatus

**Description:** Set if mute the microphone

**Format:**

C++: void SetMicMuteStatus(BOOL bMute)  
.NET: void SetMicMuteStatus(uint bMute)  
OCX: void SetMicMuteStatus(long bMute)  
DLL: void GTAPI\_SetMicMuteStatus(BOOL bMute)

**Parameters:**

bMute: 1 = Mute the microphone    0 = unmute the microphone

**Return:**

void

**Sample code:**

```
//Mute the microphone
env.SetMicMuteStatus(1);
```

### 3.18.13 GetMicMuteStatus

**Description:** Get the microphone's mute status.

**Format:**

C++: BOOL GetMicMuteStatus();  
.NET: uint GetMicMuteStatus();  
OCX: long GetMicMuteStatus();  
DLL: BOOL GTAPI\_GetMicMuteStatus();

**Parameters:**

null

**Return:**

1 = microphone is muted  
0 = microphone is in normal status

**Sample code:**

```
if(env.GetMicMuteStatus() == 1) { // Microphone is in mute status }
```

### 3.18.14    **GetSoundDeviceCount**    **deprecated**

**Description:** Get total number of sound devices in local computer. This function is deprecated. Please use GetRenderDeviceCount or GetCaptureDeviceCount instead of.

**Format:**

C++: unsigned long GetSoundDeviceCount()  
.NET: unsigned long GetSoundDeviceCount()  
OCX: unsigned long GetSoundDeviceCount()  
DLL: unsigned long GTAPI\_GetSoundDeviceCount()

**Parameters:**

**Return:**

Number of sound devices

**Sample code:**

```
//Get total number of sound devices
env.GetSoundDeviceCount();
```

### 3.18.15    **GetSoundDeviceName**    **deprecated**

**Description:** Get the name of sound device. This function is deprecated. Please use GetRenderDeviceName or GetCaptureDeviceName to instead of.

**Format:**

C++: const char\* GetSoundDeviceName(int idx)  
.NET: string GetSoundDeviceName(int idx)  
OCX: BSTR GetSoundDeviceName(int idx)  
DLL: const char\* GTAPI\_GetSoundDeviceName(int idx)

**Parameters:**

idx: index of the sound device

**Return:**

name of the sound device

**Sample code:**

```
//Get the first device name
env.GetSoundDeviceName(0);
```



### 3.18.16 GetRenderDeviceCount

**Description:** Get total number of sound render devices in local computer. Render devices are sound cards' speakers usually.

**Format:**

C++: unsigned int GetRenderDeviceCount ()  
.NET: unsigned int GetRenderDeviceCount ()  
OCX: unsigned int GetRenderDeviceCount ()  
DLL: unsigned int GTAPI\_GetRenderDeviceCount()

**Parameters:**

**Return:**

Number of render devices

**Sample code:**

```
//Get total number of speakers
env.GetRenderDeviceCount();
```

### 3.18.17 GetRenderDeviceName

**Description:** Get the name of render sound device.

**Format:**

C++: const char\* GetRenderDeviceName(int idx)  
.NET: string GetRenderDeviceName(int idx)  
OCX: BSTR GetRenderDeviceName(int idx)  
DLL: const char\* GTAPI\_GetRenderDeviceName(int idx)

**Parameters:**

idx: index of the render device

**Return:**

name of the render device

**Sample code:**

```
//Get the first render device name
env.GetRenderDeviceName(0);
```

### 3.18.18 IsRenderDevicePrimary

**Description:** Check if the render device is system primary(default) render device.

**Format:**

C++: BOOL IsRenderDevicePrimary(int idx)  
.NET: bool IsRenderDevicePrimary(int idx)  
OCX: OLEBoolean IsRenderDevicePrimary(int idx)  
DLL: BOOL GTAPI\_IsRenderDevicePrimary(int idx)

**Parameters:**

idx: index of the render device

**Return:**

Boolean to indicate if the device is primary render device.

**Sample code:**

### 3.18.19 GetCaptureDeviceCount

**Description:** Get total number of sound capture devices in local computer. Capture devices are sound cards' microphones usually.

**Format:**

C++: unsigned int GetCaptureDeviceCount()  
.NET: unsigned int GetCaptureDeviceCount()  
OCX: unsigned int GetCaptureDeviceCount()  
DLL: unsigned int GTAPI\_GetCaptureDeviceCount()

**Parameters:**

**Return:**

Number of capture devices

**Sample code:**

```
//Get total number of microphones  
env.GetCaptureDeviceCount();
```

### 3.18.20 GetCaptureDeviceName

**Description:** Get the name of capture sound device.

**Format:**

C++: `const char* GetCaptureDeviceName(int idx)`  
.NET: `string GetCaptureDeviceName(int idx)`  
OCX: `BSTR GetCaptureDeviceName(int idx)`  
DLL: `const char* GTAPI_GetCaptureDeviceName(int idx)`

**Parameters:**

idx: index of the capture device

**Return:**

name of the capture device

**Sample code:**

```
//Get the first capture device name
env.GetCaptureDeviceName(0);
```

### 3.18.21 IsCaptureDevicePrimary

**Description:** Check if the capture device is system primary(default) capture device.

**Format:**

C++: `BOOL IsCaptureDevicePrimary(int idx)`  
.NET: `bool IsCaptureDevicePrimary(int idx)`  
OCX: `OLEBoolean IsCaptureDevicePrimary(int idx)`  
DLL: `BOOL GTAPI_IsCaptureDevicePrimary(int idx)`

**Parameters:**

idx: index of the capture device

**Return:**

Boolean to indicate if the device is primary capture device.

**Sample code:**

## 3.19 Network Functions

### 3.19.1 GetLocalIPCount

**Description:** Get total number of IP addresses. Most machines only have one IP address, but for multi-home machines, they may have more than one IP address. This functions will tell you how many IP addresses the machine has.

**Format:**

C++: int GetLocalIPCount()  
.NET: int GetLocalIPCount()  
OCX: int GetLocalIPCount()  
DLL: int GTAPI\_GetLocalIPCount()

**Parameters:**

**Return:**

Number of IP addresses

**Sample code:**

```
//Get total number of IP addresses
env.GetLocalIPCount();
```

### 3.19.2 GetLocalIP

**Description:** Get local IP address.

**Format:**

C++: const char\* GetLocalIP(int idx)  
.NET: string GetLocalIP(int idx)  
OCX: BSTR GetLocalIP(int idx)  
DLL: const char\* GTAPI\_GetLocalIP(int idx)

**Parameters:**

Idx: Index of IP Address, based on 0.

**Return:**

String of IP address.

**Sample code:**

```
//Get IP addresses
env.GetLocalIP();
```

### 3.19.3 GetLocalNetworkIPAddress **deprecated**

**Description:** Get the first local IP address. Same as GetLocalIP(0).

**Format:**

```
C++: const char* GetLocalNetworkIPAddress()
.NET: string GetLocalNetworkIPAddress()
OCX: BSTR GetLocalNetworkIPAddress()
DLL: const char* GTAPI_GetLocalNetworkIPAddress()
```

**Parameters:****Return:**

String of first IP address.

**Sample code:**

```
//Get first IP addresses
env.GetLocalNetworkIPAddress();
```

### 3.19.4 GetSIPAddressInfo

**Description:** This function helps your decode SIP address string. For example, if you get a SIP address string like “Myname<sip:1234@pcbest.net>” in On\_RecvOffered event, you may need to know the caller id number in SIP string. This function can help you decode it.

**Format:**

```
C++: GT_BOOL GetSIPAddressInfo(char *sipAddr, char** pstr_Name, char**
pstr_Address, char** pstr_Port)
.NET: string GetSIPAddressInfo(int flag, string sipAddr)
OCX: BSTR GetSIPAddressInfo(int flag, BSTR sipAddr)
DLL: const char* GTAPI_GetSIPAddressInfo(int flag, const char* sipAddr)
```

**Parameters:**

sipAddr: full sip address string  
flag: 0 = display name, 1 = username, 2 = domain, 3 = port

**Return:**

Address string.

**Sample code:**

```
//Get username of sip address string, result is "1234".
const char* env.GetSIPAddressInfo(1, "<sip:1234@pcbest.net>");
```

### 3.19.5 GetDetectedNATType

**Description:** This function returns local network type detected. Refer stun protocol for network type definition.

**Format:**

```
C++: int GetDetectedNATType()
.NET: int GetDetectedNATType()
OCX: int GetDetectedNATType()
DLL: int GTAPI_GetDetectedNATType()
```

**Parameters:****Return:**

-1 = unknown, detected error  
 0 = open  
 2 = Independent Mapping, Independent Filter  
 4 = Independent Mapping, Address Dependent Filter  
 6 = Independent Mapping, Port Dependent Filter  
 8 = Dependent Mapping  
 10 = Firewalled  
 12 = Blocked or could not reach STUN server  
 14 = Unknown NAT type

**Sample code:**

```
//prompt that user may not be able to make Internet call because of the network
limitation
env.StartServer();
if(env.GetDetectedNATType() < 0 || env.GetDetectedNATType() >= 8)
{
    MessageBox("You may not be able to make Internet calls because your network
is not configured right, or firewall blocked.");
}
```

### 3.19.6 GetMappedPublicSIIPAddress

**Description:** This function gets the real public Internet IP address for SIP communication.

**Format:**

C++: `const char* GetMappedPublicSIIPAddress()`  
.NET: `string GetMappedPublicSIIPAddress()`  
OCX: `BSTR GetMappedPublicSIIPAddress()`  
DLL: `const char* GTAPI_GetMappedPublicSIIPAddress()`

**Parameters:**

**Return:**

IP address string.

**Sample code:**

```
//get ip address of sip
env.StartServer();
env.GetMappedPublicSIIPAddress();
env.GetMappedPublicSIIPPort();
```

### 3.19.7 GetMappedPublicSIIPPort

**Description:** This function gets the real public Internet IP port for SIP communication.

**Format:**

C++: `unsigned short GetMappedPublicSIIPPort()`  
.NET: `unsigned short GetMappedPublicSIIPPort()`  
OCX: `unsigned short GetMappedPublicSIIPPort()`  
DLL: `unsigned short GTAPI_GetMappedPublicSIIPPort()`

**Parameters:**

**Return:**

Port number

**Sample code:**

```
//get ip address of sip
env.StartServer();
env.GetMappedPublicSIIPAddress();
env.GetMappedPublicSIIPPort();
```

### 3.19.8 GetLocalSIPPort

**Description:** This function gets the local SIP port number. You can set local SIP port by using tag “gtsrv.sip.ip.port”.

**Format:**

C++:	unsigned short	GetLocalSIPPort ()
.NET:	unsigned short	GetLocalSIPPort ()
OCX:	unsigned short	GetLocalSIPPort ()
DLL:	unsigned short	GTAPI_GetLocalSIPPort ()

**Parameters:**

**Return:**

Port number

**Sample code:**

### 3.19.9 GetLocalRTPPort

**Description:** This function gets the local SIP port number. You can set local RTP port by using tag “gtsrv.sip.rtpstartrange”.

**Format:**

C++:	unsigned short	GetLocalRTPPort ()
.NET:	unsigned short	GetLocalRTPPort ()
OCX:	unsigned short	GetLocalRTPPort ()
DLL:	unsigned short	GTAPI_GetLocalRTPPort()

**Parameters:**

**Return:**

Port number

**Sample code:**

### 3.19.10 GetPeerSIPIPAddress

**Description:** After the call is connected, this function can be used to get the peer SIP address.



**Format:**

C++: `const char* GetPeerSIIPAddress(int ch);`  
.NET: `string GetPeerSIIPAddress(int ch);`  
OCX: `BSTR GetPeerSIIPAddress(int ch);`  
DLL: `const char* GTAPI_GetPeerSIIPAddress(int ch);`

**Parameters:**

ch: Channel Index based on 0.

**Return:**

Peer SIP address.

**Sample code:**

```
//Get Peer SIP Address after call is connected
void On_RecvConnected(int ch)
{
    printf("Peer sip address is %s", env.GetPeerSIIPAddress(ch));
}
```

### 3.19.11 GetPeerSIIPPort

**Description:** After the call is connected, this function can be used to get the peer SIP port.

**Format:**

C++: `unsigned short GetPeerSIIPPort(int ch);`  
.NET: `unsigned short GetPeerSIIPPort(int ch);`  
OCX: `unsigned short GetPeerSIIPPort(int ch);`  
DLL: `unsigned short GTAPI_GetPeerSIIPPort(int ch);`

**Parameters:**

ch: Channel Index based on 0.

**Return:**

Peer SIP port

**Sample code:**

```
//Get Peer SIP port after call is connected
void On_RecvConnected(int ch)
{
    printf("Peer sip port is %d", env.GetPeerSIIPPort(ch));
}
```

### 3.19.12 GetPeerRTPIPAddress

**Description:** After the call is connected, this function can be used to get the peer RTP address.

**Format:**

C++: `const char* GetPeerRTPIPAddress(int ch);`  
.NET: `string GetPeerRTPIPAddress(int ch);`  
OCX: `BSTR GetPeerRTPIPAddress(int ch);`  
DLL: `const char* GTAPI_GetPeerRTPIPAddress(int ch);`

**Parameters:**

ch: Channel Index based on 0.

**Return:**

Peer RTP address.

**Sample code:**

```
//Get Peer RTP Address after call is connected
void On_RecvConnected(int ch)
{
    printf("Peer RTP address is %s", env.GetPeerRTPIPAddress(ch));
}
```

### 3.19.13 GetPeerRTPIPPort

**Description:** After the call is connected, this function can be used to get the peer RTP port.

**Format:**

C++: `unsigned short GetPeerRTPIPPort(int ch);`  
.NET: `unsigned short GetPeerRTPIPPort(int ch);`  
OCX: `unsigned short GetPeerRTPIPPort(int ch);`  
DLL: `unsigned short GTAPI_GetPeerRTPIPPort(int ch);`

**Parameters:**

ch: Channel Index based on 0.

**Return:**

Peer RTP port

**Sample code:**

```
//Get Peer RTP port after call is connected
```

```
void On_RecvConnected(int ch)
{
    printf("Peer rtp port is %d", env.GetPeerRTPIPPort(ch));
}
```

### 3.19.14 GetPeerSIPContactAddress

**Description:** After the call is connected, this function can be used to get the peer SIP contact address.

**Format:**

C++: `const char* GetPeerSIPContactAddress(int ch);`  
.NET: `string GetPeerSIPContactAddress(int ch);`  
OCX: `BSTR GetPeerSIPContactAddress(int ch);`  
DLL: `const char* GTAPI_GetPeerSIPContactAddress(int ch);`

**Parameters:**

ch: Channel Index based on 0.

**Return:**

Peer SIP contact address.

**Sample code:**

```
//Get Peer SIP contact Address after call is connected
void On_RecvConnected(int ch)
{
    printf("Peer SIP contact address is %s",
env.GetPeerSIPContactAddress(ch));
}
```

## 3.20 Error Event

Error event are used to notify application level errors.

### 3.20.1 On\_RecvError

**Description:** An error occurred. For example, when channel is not idle(in a call), and you are trying to make a call on this channel. You will receive this event. Or if the channel is idle, but you are trying to play an audio on this channel, or try to transfer a call on this channel.

**Format:**

C++: void On\_RecvError(int ch, int errCode)  
.NET: void On\_RecvError(int ch, int errCode)  
OCX: void OnRecvError(int ch, int errCode)  
DLL: void GTAPI\_On\_RecvError(int ch, int errCode)

**Parameters:**

ch: Channel Index based on 0.  
errCode: Error code

100000 = GT\_ERR\_SERVER\_FUNC\_NOT\_AVAILABLE  
100001 = GT\_ERR\_CHANNEL\_INVALID  
100002 = GT\_ERR\_INVALID\_ARG  
100003 = GT\_ERR\_INVALID\_FILE\_NAME  
100011 = GT\_ERR\_CHANNEL\_NOT\_IN\_IDLE  
100012 = GT\_ERR\_CHANNEL\_NOT\_IN\_CONNECTED  
100013 = GT\_ERR\_CHANNEL\_NOT\_IN\_OFFERED  
100018 = GT\_ERR\_CHANNEL\_AUDIO\_RES\_BUSY  
100021 = GT\_ERR\_SETUP\_CALL\_FAILED  
100022 = GT\_ERR\_ANSWER\_CALL\_FAILED  
100023 = GT\_ERR\_HOLD\_CALL\_FAILED  
100024 = GT\_ERR\_TRANSFER\_CALL\_FAILED  
100025 = GT\_ERR\_ACCEPT\_CALL\_FAILED  
100026 = GT\_ERR\_RING\_CALL\_FAILED  
100031 = GT\_ERR\_PLAY\_DTMF\_STR\_FAILED  
100032 = GT\_ERR\_START\_DTMF\_DETECTION\_FAILED  
100033 = GT\_ERR\_STOP\_DTMF\_DETECTION\_FAILED  
100034 = GT\_ERR\_START\_DX\_AUDIO\_FAILED  
100035 = GT\_ERR\_STOP\_DX\_AUDIO\_FAILED  
100036 = GT\_ERR\_AUDIO\_FILENAME\_IS\_NULL  
100037 = GT\_ERR\_PLAY\_AUDIO\_FAILED  
100038 = GT\_ERR\_RECORD\_AUDIO\_FAILED  
100039 = GT\_ERR\_TX\_FAX\_FAILED  
100040 = GT\_ERR\_RX\_FAX\_FAILED  
100041 = GT\_ERR\_RESET\_DX\_AUDIO\_FAILED  
100042 = GT\_ERR\_START\_MUSIC\_ON\_HOLD  
100043 = GT\_ERR\_STOP\_MUSIC\_ON\_HOLD

**Return:****Sample code:**

```
//log out
void On_RecvError(int ch, int errCode)
{
    printf("Error %d", errCode);
}
```

## 3.21 Timer functions

### 3.21.1 StartTimer

**Description:** Start timer on the channel. Later on, an event On\_Timer will be triggered once the timer expired.

**Format:**

C++: void StartTimer(int ch, unsigned long milli\_secs)  
.NET: void StartTimer(int ch, unsigned long milli\_secs)  
OCX: void StartTimer(int ch, unsigned long milli\_secs)  
DLL: void GTAPI\_StartTimer(int ch, unsigned long milli\_secs)

**Parameters:**

ch: Channel Index based on 0.  
milli\_secs: in milliseconds to expire.

**Return:**

**Sample code:**

```
//start 2 second timer on channel 0  
env.StartTimer(0, 2000);
```

### 3.21.2 StopTimer

**Description:** Stop timer on the channel.

**Format:**

C++: void StopTimer(int ch)  
.NET: void StopTimer(int ch)  
OCX: void StopTimer(int ch)  
DLL: void GTAPI\_StopTimer(int ch)

**Parameters:**

ch: Channel Index based on 0.

**Return:**

**Sample code:**

```
//stop timer on channel 0
```

```
env.StopTimer(0);
```

### 3.21.3 OnTimer

**Description:** Event to notify that channel timer expired.

**Format:**

```
C++: void OnTimer(int ch)
.NET: void OnTimer(int ch)
OCX: void OnTimer(int ch)
DLL: void GTAPI_SetCB_OnTimer
```

**Parameters:**

ch: Channel Index based on 0.

**Return:**

**Sample code:**

```
//Restart Timer when Timer expired
void On_Timer(int ch)
{
    StartTimer(ch, 2000);
}
```

## 3.22 SDK License Functions

There are several configuration tags for license. You **DO NOT** have to set them.

**"gtsrv.lic.mac.addr"** You set this tag if you want your license key is tied to a specific MAC address. Defaultly SDK will choose the first MAC.

**"gtsrv.lic.usb.key.driver"** The USB key you want the license key is tied to. It is the driver letter, like "E:", or "G:".

**"gtsrv.lic.file.dir"** You set this tag if you want your license file to be saved into a specific address. Defaultly SDK will put it into c:\Windows folder.

**"gtsrv.lic.file.name"** The license file name to be saved. If you don't set, it will be a file name like this: A0567C.....hex

### 3.22.1 SetAppName

**Description:** Set your application name. PCBest will send you your application name once you purchased the license.

**Format:**

C++: void SetAppName(const char\* s)  
.NET: void SetAppName(string s);  
OCX: void SetAppName(BSTR s);  
DLL: void GTAPI\_SetAppName(const char\* s);

**Parameters:**

s: application name string

**Return:**

**Sample code:**

```
//Route to set license information
env.CreateEnv(); //ONLY .NET, OCX, and DLL need to use this function
env.CFG_SetValue(..., ...) //other configurations.
env.SetAppName("my app name");
env.CFG_SetValue("gtsrv.licence.key", "xxxx-xxxx-xxxx-...."); //set license
key
env.StartServer();
if(env.GetLicTo() != "")
{
    printf("License to %s", env.GetLicTo());
}
else
{
    MessageBox("ERROR, Not Licensed Software!");
}
```

### 3.22.2 GetLicTo

**Description:** Get licensed info. (Usually it is your company name, or your name)

**Format:**

C++: const char\* GetLicTo()  
.NET: string GetLicTo()  
OCX: BSTR GetLicTo()  
DLL: const char\* GTAPI\_GetLicTo()

**Parameters:**

**Return:**

Licensed name string.

**Sample code:**

```
//Route to set license information
env.CreateEnv(); //ONLY .NET, OCX, and DLL need to use this function
env.CFG_SetValue(..., ...) //other configurations.
env.SetAppName("my app name");
env.CFG_SetValue("gtsrv.licence.key", "xxxx-xxxx-xxxx-...."); //set license
key
env.StartServer();
if(env.GetLicTo() != "")
{
    printf("License to %s", env.GetLicTo());
}
else
{
    MessageBox("ERROR, Not Licensed Software!");
}
```

### 3.22.3 Tags for license

**“gtsrv.lic.file.dir”**

Default the SDK will write the license validation file into c:\Windows. If you want the license file to save into other folder, please set this tag. Sample:

```
CFG_SetValue("gtsrv.lic.file.dir", "C:\\"); //Set it to C:\
```

**“gtsrv.lic.mac.addr”**

Choose a MAC address to bind with the license key. If you have multiple NIC in one machine, you can choose one. If you don't set, the SDK will use the first available NIC. Sometimes if the machine has a wireless NIC, it may be disabled. You'd better choose a wired Ethernet card's MAC address to bound with the license key. Like this:

```
CFG_SetValue("gtsrv.lic.mac.addr", "00-6F-4E-67-91-A0");
```

**"gtsrv.lic.usb.key.driver"**

Sometimes you may want your license key to bind with a USB driver. Please set this tag instead of using above MAC address. You will need to prepare a free USB driver key always be able to plug into the machine for SDK to validate. This has advantage when machine is broken or need to be changed, you just need to unplug your key, and plug it into another machine, then your app can run.



## 3.23 Human voice or Answering Machine Detection

Please set "gtsrv.human.detect.enabled" and "gtsrv.sip.on.in.vad" both to 1, to enable this feature.

There are two thresholds,

"gtsrv.human.detect.duration" means how many milliseconds totally to detect the human or machine after call is connected. Defaultly it is 4000 (4 seconds)

"gtsrv.human.detect.voiceon" means milliseconds threshold for first sentence length. If the first sentence is less than this threshold, it is considered as a human voice, otherwise it is machine. defaultly it is 2000 (2 seconds)

### 3.23.1 On\_DetectHumanVoiceDone

**Description:** Event to indicate that detecting human voice is done

**Format:**

C++: void On\_DetectHumanVoiceDone(int ch, int result)

.NET: void On\_DetectHumanVoiceDone(int ch, int result)

OCX: void OnDetectHumanVoiceDone(int ch, int result)

DLL: void GTAPI\_SetCB\_On\_DetectHumanVoiceDone

**Parameters:**

ch: Channel Index based on 0.

result:

0 = Answering Machine

1 = Human voice

-1 = silence (no voice at all in the "gtsrv.human.detect.duration" milliseconds.)

-2 = detected voice, but unknown because "gtsrv.human.detect.duration" is reached.

**Return:**

**Sample code:**

## 3.24 Other functions

### 3.24.1 GetChanAudioCodec

**Description:** Get audio codec code that channel is using. Only use this function when the call is connected.

**Format:**

C++: int GetChanAudioCodec(int ch)  
.NET: int GetChanAudioCodec(int ch)  
OCX: int GetChanAudioCodec(int ch)  
DLL: int GTAPI\_GetChanAudioCodec(int ch)

**Parameters:**

ch: Channel Index based on 0.

**Return:**

0 = MULAW  
8 = ALAW  
3 = GSM  
18 = G729  
98 = G726-32  
102 = Speex  
101 = iLBC 30ms  
100 = iLBC 20ms

**Sample code:**

```
//Log out
void On_RecvConnected(int ch)
{
    printf("Audio Codec is using %d", GetChanAudioCodec(ch));
}
```

### 3.24.2 GetChanLastMsgCode

**Description:** Get channel last response message code. This function can be used to retrieve the reason why last call is not succeed.

**Format:**

C++: int GetChanLastMsgCode(int ch)  
.NET: int GetChanLastMsgCode(int ch)  
OCX: int GetChanLastMsgCode(int ch)  
DLL: int GTAPI\_GetChanLastMsgCode(int ch)

**Parameters:**

ch: Channel Index based on 0.

**Return:**

SIP response code. Please refer to RFC 3261 for all SIP response code. Here is a short list:

"400" ; Bad Request

"401" ; Unauthorized  
 "402" ; Payment Required  
 "403" ; Forbidden  
 "404" ; Not Found  
 "405" ; Method Not Allowed  
 "406" ; Not Acceptable  
 "407" ; Proxy Authentication Required  
 "408" ; Request Timeout  
 "410" ; Gone  
 "413" ; Request Entity Too Large  
 "414" ; Request-URI Too Large  
 "415" ; Unsupported Media Type  
 "416" ; Unsupported URI Scheme  
 "420" ; Bad Extension  
 "421" ; Extension Required  
 "423" ; Interval Too Brief  
 "480" ; Temporarily not available  
 "481" ; Call Leg/Transaction Does Not Exist  
 "482" ; Loop Detected  
 "483" ; Too Many Hops  
 "484" ; Address Incomplete  
 "485" ; Ambiguous  
 "486" ; Busy Here  
 "487" ; Request Terminated  
 "488" ; Not Acceptable Here  
 "491" ; Request Pending  
 "493" ; Undecipherable

**Sample code:**

```

//Log out
void On_RecvIdle(int ch)
{
    if(this is an outbound call)
        if(this is not a successful call, for example, no On_RecvConnected
        event)
            printf("Call is not successful because error code is %d",
            GetChanLastMsgCode(ch));
}

```

### 3.24.3 GetChanLastMsgText

**Description:** Get channel last response message text. This function can be used to retrieve the reason why last call is not succeed.

**Format:**

C++: `const char* GetChanLastMsgText(int ch)`  
.NET: `string GetChanLastMsgText(int ch)`  
OCX: `BSTR GetChanLastMsgText(int ch)`  
DLL: `const char* GTAPI_GetChanLastMsgText(int ch)`

**Parameters:**

ch: Channel Index based on 0.

**Return:**

SIP response code. Please refer to RFC 3261 for all SIP response code.

**Sample code:**

```
//Log out
void On_RecvIdle(int ch)
{
    if(this is an outbound call)
        if(this is not a successful call, for example, no On_RecvConnected
        event)
            printf("Call is not successful because of %s",
            GetChanLastMsgText(ch));
}
```

### 3.24.4 SetChanUserData

**Description:** Associate a user data to a channel. Sometimes the application may need to save some data for the channel. This function allows the programmer to save 10 unsigned long data to a specific channel.

**Format:**

C++: `bool SetChanUserData(int chan_id, int data_idx, unsigned long d)`  
.NET: `bool SetChanUserData(int chan_id, int data_idx, ulong d)`  
OCX: `short SetChanUserData(int chan_id, int data_idx, ulong d)`  
DLL: `bool GTAPI_SetChanUserData(int chan_id, int data_idx, ulong d)`

**Parameters:**

ch: Channel Index based on 0.  
data\_idx: can be 0 – 9.  
d: data value

**Return:**

Boolean: if it is set successfully.

**Sample code:**

### 3.24.5 GetChanUserData

**Description:** Retrieve the data which is set by SetChanUserData.

**Format:**

C++: `bool GetChanUserData(int chan_id, int data_idx, unsigned long* d)`  
.NET: `bool GetChanUserData(int chan_id, int data_idx, UIntPtr d)`  
OCX: `short GetChanUserData(int chan_id, int data_idx, ulong* d)`  
DLL: `bool GTAPI_GetChanUserData(int chan_id, int data_idx, unsigned long* d)`

**Parameters:**

ch: Channel Index based on 0.  
data\_idx: can be 0 – 9.  
d: pointer to an unsigned long type

**Return:**

Boolean: if the function is successful.

**Sample code:**

### 3.24.6 SetMainWnd

**Description:** Set a Windows Handle to env class. Env class will use this Wnd handle for microphone and speaker control, or for Windows message. You don't have to set a Windows handle if you are running a server application, or if there is no sound card in the system.

**Format:**

C++: `void SetMainWnd(HWND wnd)`  
.NET: `void SetMainWnd(HWND wnd)`  
OCX: `void SetMainWnd(HWND wnd)`  
DLL: `void GTAPI_SetMainWnd(HWND wnd)`

**Parameters:**

wnd: Windows Handle.

**Return:**

**Sample code:**

```
//Set Windows Handle
env.CreateEnv(); //ONLY DLL, OCX and .NET developers need this one
```

```
env.SetMainWnd(GetWndHandle());
env.CFG_SetValue(..., ...) //configuration items
env.StartServer();
```

### 3.24.7 GetTotalChannelNumber

**Description:** Get total channel count.

**Format:**

```
C++: unsigned int GetTotalChannelNumber()
.NET: unsigned int GetChannelCount()
OCX: unsigned int GetChannelCount ()
DLL: unsigned int GTAPI_GetChannelCount()
```

**Parameters:**

**Return:**

Totoal channel number

**Sample code:**

```
//log out
env.CreateEnv(); //ONLY DLL, OCX and .NET developers need this one
env.CFG_SetValue(..., ...) //configuration items
env.StartServer();
int cnt = env.GetChannelCount();
```

### 3.24.8 GetChanAudioRecordFileName

**Description:** Get current audio recording file name(When set tag "gtphone.audio.record.enabled" to 1). Note: please using this function after the call is connected, or call is just idle. You only can operate(move, rename, or mp3 process) the file after the call is idle.

**Format:**

```
C++: const char* GetChanAudioRecordFileName(int ch);
.NET: string GetChanAudioRecordFileName(int ch);
OCX: BSTR GetChanAudioRecordFileName(int ch);
DLL: const char* GTAPI_GetChanAudioRecordFileName(int ch);
```

**Parameters:**

ch: Channel Index

**Return:**

String of file name

**Sample code:**

```
void On_RecvConnected(int ch)
{
    Send_StartDXAudio(ch);
    GetChanAudioRecordFileName(ch);
}

or

void On_RecvIdle(int ch)
{
    const char* srcFileName = GetChanAudioRecordFileName(ch);
    MoveFile(srcFileName, destFileName);
}
```

### 3.24.9 SetChanCallExtraSIPHeader

**Description:** Set extra SIP header in SIP INVITE when making out a call. It support multiple SIP headers by using ‘\n’ in the string. **Note:** You have to call this function before using Send\_Make to call out for every call.

**Format:**

C++: void SetChanCallExtraSIPHeader(int ch, const char\* s);  
.NET: void SetChanCallExtraSIPHeader(int ch, string s);  
OCX: void SetChanCallExtraSIPHeader(int ch, BSTR s);  
DLL: void GTAPI\_SetChanCallExtraSIPHeader (int ch, const char\* s);

**Parameters:**

ch: Channel Index  
s: Extra SIP header

**Return:**

null

**Sample code:**

```
SetChanCallExtraSIPHeader(0, "Billing-Code: 123");
SetChanCallExtraSIPHeader(0, "Billing-Code: 123\nHome-Addr: 87 Bank Street");
```

### 3.24.10 SetChanAudioLevel

**Description:** Control channel's volume of sound.

**Format:**

C++: void SetChanAudioLevel(int ch, float inLevel, float outLevel);  
.NET: void SetChanAudioLevel(int ch, float inLevel, float outLevel);  
OCX: void SetChanAudioLevel(int ch, float inLevel, float outLevel);  
DLL: void GTAPI\_SetChanAudioLevel(int ch, float inLevel, float outLevel);

**Parameters:**

ch: Channel Index

inLevel: inbound sound level. defaultly it is 1.00, so there is no any change.

outlevel: outbound sound level. defaultly it is 1.00, so there is no any change.

**Return:**

null

**Sample code:**

```
SetChanAudioLevel(0, 2.0, 2.0);
```

### 3.24.11 On\_SIPMsg

**Description:** An event for SIP raw messages. NOTE: IN ORDER TO TRIGGER THIS EVENT, YOU NEED TO SET "gtsrv.sip.on.msg" TO "1".

**Format:**

C++: void On\_SIPMsg(int dir, const char\* src\_ip, unsigned short src\_port, const char\* dest\_ip, unsigned short dest\_port, char\* buf, int len)  
.NET: void On\_SIPMsg(int dir, string src\_ip, unsigned short src\_port, string dest\_ip, unsigned short dest\_port, string buf, int len)  
OCX: void On\_SIPMsg(int dir, BSTR src\_ip, unsigned short src\_port, BSTR dest\_ip, unsigned short dest\_port, BSTR buf, int len)  
DLL: void GTAPI\_SetChanAudioLevel(int ch, float inLevel, float outLevel);

**Parameters:**

dir: 0 = incoming, 1 = outgoing

src\_ip: message from ip address

src\_port: message from ip port

dest\_ip: message to ip address

dest\_port: message to ip port



buf: sip message buffer  
 len: message length in bytes

**Return:**  
 null

## 4 Proxy API

PCBest SIP SDK supports a set of proxy API to help programmers to build SIP Proxy Server Applications, like SIP registration Server, SIP PBX, SIP Balance Server.

PCBest Networks also developed a Commercial SIP PBX(with free edition) based on SIP SDK. You are welcome to use these interfaces to develop your own logic SIP PBX applications. Also, if you want a fast start for your PBX project, please feel free to contact us about your need.

### 4.1 Initialize Proxy Site

A proxy site is a proxy server which can have a group of users. PCBest SIP SDK allows you to set more than one proxy site. Please set the following tags to create proxy sites.

“**gtsrv.sip.proxy.sites.num**”: The number of Proxy Sites

“**gtsrv.sip.proxy.site1.domain**”: The first site’s domain. Usually it is the local private ip address, and public ip address. If you have domain name mapped to your server, you can also add your domain name into this list.

“**gtsrv.sip.proxy.site1.recordroute**”: If record route when proxy calls. Please always set it to 1.

“**gtsrv.sip.proxy.site1.udp.relay**”: If do UDP replaying for calls. Please always set it to 0.

#### 4.1.1 InitProxySites

**Description:** Initialize the proxy sites. Usually this function is called when the app starts after StartServer function.

**Format:**

C++: void InitProxySites();  
.NET: void InitProxySites();  
OCX: void InitProxySites();  
DLL: void GTAPI\_InitProxySites();

**Parameters:****Return:****Sample code:**

```
GTSIPAPI1.StartServer; //You should already have this line in the code

GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.sites.num', '1'); //only one proxy site

string site1domain = GTSIPAPI1.GetMappedPublicSIPIAddress + ';' +
GTSIPAPI1.GetLocalIP(0) + ';' + 'mysipproxy.com'

GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.site1.domain', site1domain);
GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.site1.recordroute', '1');
GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.site1.udp.relay', '0');

GTSIPAPI1.InitProxySites;
```

## 4.2 Free Proxy Site

### 4.2.1 FreeProxySites

**Description:** Free Proxy Sites when app ends.

**Format:**

C++: void FreeProxySites();  
.NET: void FreeProxySites();  
OCX: void FreeProxySites();  
DLL: void GTAPI\_FreeProxySites();

**Parameters:****Return:****Sample code:**

```
//code to shut down the server
```

```
api.FreeProxySites();
api.StopServer();
```

## 4.3 Add, Delete, Change or Disable Proxy Site User

### 4.3.1 ProxySetUserInfo

**Description:** Set user info for the proxy. You can use this function to add proxy user, delete proxy user, or change the proxy user information.

**Format:**

C++: void ProxySetUserInfo(unsigned int pid, const char\* username, const char\* passwd, time\_t regt, unsigned int regex, const char\* contactaddr, **const char\* mapped\_contact**, const char\* uaname, int nattype, **const char\* src\_ip**, **unsigned short src\_port**, const char\* from\_id, const char\* to\_id);  
.NET: void ProxySetUserInfo(uint pid, string username, string passwd, uint regt, uint regex, string contactaddr, string mapped\_contact, string uaname, int nattype, string src\_ip, ushort src\_port, const char\* from\_id, const char\* to\_id);  
OCX: void ProxySetUserInfo(uint pid, BSTR username, BSTR passwd, uint regt, uint regex, BSTR contactaddr, BSTR uaname, int nattype);  
DLL: void GTAPI\_ProxySetUserInfo(unsigned int pid, const char\* username, const char\* passwd, time\_t regt, unsigned int regex, const char\* contactaddr, **const char\* mapped\_contact**, const char\* uaname, int nattype, **const char\* src\_ip**, **unsigned short src\_port**, const char\* from\_id, const char\* to\_id);

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

username: Unique user name for this user. It is usually used by client site to register on this SIP proxy server. Typically, for a PBX system, it can be 101 for extension number.

passwd: the password for this user. The client will use this password to register on this server. If this parameter is null(""), then it means to delete this user in the proxy user list.

regt: Please pass 0(not registered). or unix time(registered time).

regex: Registered expiration Seconds. 3600 = 1 hour, or 0 = unregistered.

contactaddr: This is the user's SIP contact address. Please just pass "".

mapped\_contact: mapped contact address that SDK figured out by using REGISTER source IP and port. If it is "", then org\_contact is alright, and matching.

uaname: This user's user agent name. This is usually user softphone's maker info.

Nattype: please pass -1 for now.

src\_ip: REGISTER Message source IP address

src\_port: REGISTER Message source IP port

from\_id : the original SIP FROM in Register message, usually it is something like Mike<sip:101@192.168.1.100>. It can be "" if just setting up the extension.  
 to\_id : the most of case, it is the same as from\_id. It can be "" if just setting up the extension.

**Return:**

null

**Sample code:****//Init users for this proxy site**

//assume you have an array(**users**) which holds all the sip users.

//We just need to set all users info into SDK

GTSIPAPI1.StartServer; *//You should already have this line in the code*

GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.sites.num', '1'); *//only one proxy site*

string site1domain = GTSIPAPI1.GetMappedPublicSIPIPAddress + ';' +  
 GTSIPAPI1.GetLocalIP(0) + ';' + 'mysipproxy.com'

GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.site1.domain', site1domain);

GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.site1.recordroute','1');

GTSIPAPI1.CFGSetValue('gtsrv.sip.proxy.site1.udp.relay','0');

GTSIPAPI1.InitProxySites;

For(int i=0; i<users.size; i++)

```
{
    GTSIPAPI1.ProxySetUserInfo(0, users[i].UserName, users[i].Passwd, 0,
    0, "", "", -1)
}
```

**//Delete user 101**

GTSIPAPI1.ProxySetUserInfo(0, "101", "", 0, 0, "", "", -1, "", "")

### 4.3.2 ProxyDisableAllUsers

**Description:** Disable all users in the proxy site.

**Format:**

C++: void ProxyDisableAllUsers(unsigned int pid);

.NET: void ProxyDisableAllUsers(unsigned int pid);

OCX: void ProxyDisableAllUsers(unsigned int pid);  
 DLL: void GTAPI\_ProxyDisableAllUsers(unsigned int pid);

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

**Return:**

null

**Sample code:**

### 4.3.3 ProxySetUserMsg

**Description:** Set proxy user's MWI(Message Waiting Indication).

**Format:**

C++: void ProxySetUserMsg(uint pid, const char\* username, bool bMsgWait, const char\* sMsgAcct, int newCount, int oldCount, int cnt3, int cnt4);  
 .NET: void ProxySetUserMsg(uint pid, string username, bool bMsgWait, string sMsgAcct, int newCount, int oldCount, int cnt3, int cnt4);  
 OCX: void ProxySetUserMsg(uint pid, BSTR username, bool bMsgWait, BSTR sMsgAcct, int newCount, int oldCount, int cnt3, int cnt4);  
 DLL: void GTAPI\_ProxySetUserMsg( (uint pid, const char\* username, bool bMsgWait, const char\* sMsgAcct, int newCount, int oldCount, int cnt3, int cnt4);

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

username: Proxy extension's user name, like 101, 1002,....

bMsgWait: if there is message waiting? true is yes. then extension's MWI light will turn on.

sMsgAcct: sip account for extension to call back, to retrieve messages(voice mails).

Format should be standard SIP address like <sip:123@192.168.1.100:5060>.

newCount: new message count

oldCount: old message count

**Return:**

null

**Sample code: (in C#)**

```
if(nNew > 0)
    env.ProxySetUserMsg(0, extn.UserName, true, sipVMB, nNew, nOld, 0, 0);
else
```

```
env.ProxySetUserMsg(0, extn.UserName, false, sipVMB, nNew, nOld, 0, 0);
```

### 4.3.4 ProxyUpdateUserState

**Description:** SIP BLF support for Proxy API.

**Format:**

C++: ProxyUpdateUserState(unsigned int pid, const char\* username, const char\* direction, const char\* state);

.NET: void ProxyUpdateUserState(uint pid, string username, string direction, string state);

OCX: void ProxyUpdateUserState(uint pid, BSTR username, BSTR direction, BSTR state);

DLL: void GTAPI\_ProxyUpdateUserState(uint pid, const char\* username, const char\* direction, const char\* state);

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

username: Proxy extension's user name, like 101, 1002,....

direction: can be set to "recipient", "initiator", or just "".

state: can be set to:

"confirmed" ==> connected

"early" ==> ringing

"terminated" or "void" ==> idle

**Return:**

null

**Sample code: (in C#)**

```
env.ProxyUpdateUserState(0, "101", "recipient", "early");
```

## 4.4 Events

### 4.4.1 On\_ProxyUserRegistered

**Description:** This event tells that a user just registered on the local proxy server. You should update your list of users by using this event's data. For example, update the user's contact address.

**Format:**

C++: void On\_ProxyUserRegistered(unsigned int pid, const char\* username, time\_t tnow, unsigned int exp\_sec, const char\* org\_contact, const char\* **mapped\_contact**, const char\* szUName, int UANatType, const char\* szFromID, const char\* szToID, **const char\* SrcIP, unsigned short SrcPort**)

.NET: void On\_ProxyUserRegistered(uint pid, string username, DateTime tnow, uint exp\_sec, string org\_contact, **string mapped\_contact**, string szUName, int UANatType, string szFromID, string szToID, **string SrcIP, ushort SrcPort**)

OCX: void OnProxyUserRegistered(long ProxyID, BSTR UserName, DATE RegTime, long ExpireSec, BSTR ContactAddr, BSTR UserAgentName, long UserAgentNatType, BSTR FromID, BSTR ToID)

DLL: void GTAPI\_SetCB\_OnProxyUserRegistered

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

username: User's username. It must be one of users you used ProxySetUserInfo function to set.

tnow or RegTime: Current Registration Time

exp\_sec or ExpireSec: how many seconds the client registered. If it is 0, it means the client wants to unregister.

**Org\_Contact or ContactAddr**: User's Contact Address in SIP format, like this: <sip:123@220.12.34.123:5078>. This is the original contact address showing in REGISTER message.

**Mapped\_Contact**: mapped contact address that SDK figured out by using REGISTER source IP and port. If it is "", then org\_contact is alright, and matching.

szUName: Client's User Agent Name. This string usually has maker info of client's SIP softphone or hardware phone.

UANatType: Network Type. -1 for now.

szFromID, szToID: User's SIP ID used to register.

SrcIP: REGISTER Message source IP address

SrcPort: REGISTER Message source IP port

**Return:**

null

**Sample code:**

```
void On_ProxyUserRegistered(unsigned int pid, const char* username, time_t tnow,
    unsigned int exp_sec, const char* contact, const char* szUName, int UANatType,
    const char* szFromID, const char* szToID)
```

```
{
```

```
//assume I have an array(UserList), which contains all users info.
```

```
    For(int i=0; i<UserList.Size(); i++)
```

```
    {
```

```
        if(UserList[i].UserName == username)
```

```

        {
            UserList[i].RegTime = t_now;
            UserList[i].RegExpSec= exp_sec;
            UserList[i].ContactAddress = contact;
            break;
        }
    }
}

```

#### 4.4.2 On\_ProxyNewCallSession

**Description:** This event tells that a new call coming in, and the proxy needs to deal with it according to the return string of this event.

**Format:**

C++: const char\* On\_ProxyNewCallSession(unsigned int pid, unsigned int sid, GT\_HANDLE msg, const char\* fromid, const char\* toid, const char\* suri, const char\* via, const char\* saddr, unsigned short nport, bool bCredit)  
.NET: string On\_ProxyNewCallSession(uint pid, uint sid, ulong msg, string fromid, string toid, string suri, string via, string saddr, ushort nport, bool bCredit)  
OCX: void OnProxyNewCallSession(BSTR\* ResultStr, long ProxyID, long SessionID, long MsgHandle, BSTR FromID, BSTR ToID, BSTR RequestURI, BSTR RequestVia, BSTR MsgIPAddr, long MsgIPPort, long Credit)  
DLL: void GTAPI\_SetCB\_OnProxyNewCallSession

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

sid: Call Session ID

msg: Message Handle

FromID, ToID, RequestURI, RequestVia: IDs from INVITE Message

MsgIPAddr, MsgIPPort: The ip and port of source message.

Credit: If this INVITE has right authorization(credit) to call.

**Return:**

Possible return values:

"sip:undefined" = wait until **ProxySetNewCallSessionAddr** is called

Sometimes you cannot immediately decide what to do with this call, and then you can call ProxySetNewCallSessionAddr function to provide an address later.

**NOTE: The OCX(ActiveX) users should(have to) use ProxySetNewCallSessionAddr function later to provide one of the following values:**



"**channel**" = to channel. Default value. Then one of the channels will have On\_RecvOffered event triggered.

"**unauthorized**" = need credit to process. You should check the fromID, to see if it is a call from proxy site's user(or extension of PBX). If it does, and Credit is false, then "unauthorized" string is returned to ask authorization to call out.

"**sip:123@abc.com**" = with right credit or don't need credit, processed by proxy site and transfered to this address

"|" can be used to separate the credit info. For example:

"To<sip:123@abc.com>|From<sip:456@abc.com>|User<1234>|Password<4567>"

Otherwise, if none of above matched, refer to user's contact info.

### Sample code:

```
const char* On_ProxyNewCallSession(unsigned int pid, unsigned int sid, GT_HANDLE
msg, const char* fromid, const char* toid, const char* suri, const char* via, const char*
saddr, unsigned short nport, bool bCredit)
{
    return "channel"; //let the channel to handle this call
}
```

## 4.4.3 On\_ProxyCallSessionStatus

**Description:** This event reports the call session status.

### Format:

C++: void On\_ProxyCallSessionStatus(unsigned int pid, unsigned int sid, time\_t sbegin, time\_t send, int status, const char\* szFrom, const char\* szTo, const char\* szURI, const char\* szVia, const char\* szRealContact)

.NET: void On\_ProxyCallSessionStatus(uint pid, uint sid, time\_t sbegin, time\_t send, int status, const char\* szFrom, const char\* szTo, const char\* szURI, const char\* szVia, const char\* szRealContact)

OCX: void OnProxyCallSessionStatus(long ProxyID, long SessionID, DATE BeginTime, DATE EndTime, long Status, BSTR FromID, BSTR ToID, BSTR RequestURI, BSTR RequestVia, BSTR RealContactAddr)

DLL: void GTAPI\_SetCB\_OnProxyCallSessionStatus

### Parameters:

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

sid: Call Session ID.

sbegin, send: Call Session Start and End Time

status: Call Session Status.

0 = GT\_SIP\_PROXY\_SESSION\_NULL,

1 = GT\_SIP\_PROXY\_RECV\_ORIG\_INVITE,  
 2 = GT\_SIP\_PROXY\_SENT\_ORIG\_TRY,  
 3 = GT\_SIP\_PROXY\_FRWD\_ORIG\_INVITE\_TO\_TERM,  
 4 = GT\_SIP\_PROXY\_RECV\_TERM\_TRY,  
 5 = GT\_SIP\_PROXY\_RECV\_TERM\_AUTH\_REQUIRED,  
 6 = GT\_SIP\_PROXY\_RECV\_TERM\_RING,  
 7 = GT\_SIP\_PROXY\_FRWD\_TERM\_RING\_TO\_ORIG,  
 8 = GT\_SIP\_PROXY\_RECV\_TERM\_FINAL\_RESP,  
 9 = GT\_SIP\_PROXY\_FRWD\_TERM\_FINAL\_RESP\_TO\_ORIG,  
 10 = GT\_SIP\_PROXY\_RECV\_ORIG\_ACK,  
 11 = GT\_SIP\_PROXY\_FRWD\_ORIG\_ACK\_TO\_TERM,  
 12 = GT\_SIP\_PROXY\_SESSION\_CONNECTED,  
 13 = GT\_SIP\_PROXY\_RECV\_REINVITE,  
 14 = GT\_SIP\_PROXY\_FRWD\_REINVITE,  
 15 = GT\_SIP\_PROXY\_RECV\_REINVITE\_RESP,  
 16 = GT\_SIP\_PROXY\_FRWD\_REINVITE\_RESP,  
 17 = GT\_SIP\_PROXY\_RECV\_REINVITE\_ACK,  
 18 = GT\_SIP\_PROXY\_FRWD\_REINVITE\_ACK,  
 19 = GT\_SIP\_PROXY\_RECV\_REQUEST,  
 20 = GT\_SIP\_PROXY\_FRWD\_REQUEST,  
 21 = GT\_SIP\_PROXY\_RECV\_RESP,  
 22 = GT\_SIP\_PROXY\_FRWD\_RESP,  
 23 = GT\_SIP\_PROXY\_RECV\_BYE,  
 24 = GT\_SIP\_PROXY\_FRWD\_BYE,  
 25 = GT\_SIP\_PROXY\_RECV\_BYE\_OK,  
 26 = GT\_SIP\_PROXY\_FRWD\_BYE\_OK,  
 27 = GT\_SIP\_PROXY\_RECV\_ORIG\_CANCEL,  
 28 = GT\_SIP\_PROXY\_SENT\_ORIG\_CANCEL\_RESP\_TO\_ORIG,  
 29 = GT\_SIP\_PROXY\_FRWD\_ORIG\_CANCEL\_TO\_TERM,  
 30 = GT\_SIP\_PROXY\_RECV\_TERM\_RESP\_TO\_CANCEL,  
 31 = GT\_SIP\_PROXY\_SENT\_TERM\_ACK\_OF\_CANCEL,  
 32 = GT\_SIP\_PROXY\_TRANS\_VMB\_SENT\_CANCEL\_TO\_TERM,  
 33 = GT\_SIP\_PROXY\_TRANS\_VMB\_GET\_CANCEL\_RESP\_FROM\_TERM,  
 34 =  
 GT\_SIP\_PROXY\_TRANS\_VMB\_WAITING\_FOR\_INVITING\_VMB\_ADDRESS,  
 35 = GT\_SIP\_PROXY\_WAITING\_FOR\_TERM\_ADDR\_FROM\_APP,  
 36 = GT\_SIP\_PROXY\_CONNECTION\_STEP1,  
 37 = GT\_SIP\_PROXY\_CONNECTION\_DONE,

FromID, ToID, RequestID, ViaAddr: Call's Info

RealContactAddr: The real final address this call is trying to reach.

**Return:**

null

**Sample code:**

## 4.4.4 On\_ProxyCallTransaction

**Description:** This event tells that a call session is done.

**Format:**

C++: void On\_ProxyCallTransaction(unsigned int pid, unsigned int sid, const char \*szFrom, const char\* szTo, const char\* szUri, const char\* szVia, const char\* szRealContact, time\_t t\_start, time\_t t\_end, int session\_status, int result\_code, const char\* result\_txt)

.NET: void On\_ProxyCallTransaction(uint pid, uint sid, string szFrom, string szTo, string szUri, string szVia, string szRealContact, DateTime t\_start, DateTime t\_end, int session\_status, int result\_code, string result\_txt)

OCX: void OnProxyCallTransaction(long ProxyID, long SessionID, BSTR FromID, BSTR ToID, BSTR RequestURI, BSTR RequestVia, BSTR RealContactAddr, DATE BeginTime, DATE EndTime, long StatusCode, long ResultCode, BSTR ResultStr)

DLL: void GTAPI\_SetCB\_OnProxyCallTransaction

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

sid: Call Session ID.

sbegin, send: Call Session Start and End Time

session\_status: Call Session Status.

0 = GT\_SIP\_PROXY\_SESSION\_NULL,  
 1 = GT\_SIP\_PROXY\_RECV\_ORIG\_INVITE,  
 2 = GT\_SIP\_PROXY\_SENT\_ORIG\_TRY,  
 3 = GT\_SIP\_PROXY\_FRWD\_ORIG\_INVITE\_TO\_TERM,  
 4 = GT\_SIP\_PROXY\_RECV\_TERM\_TRY,  
 5 = GT\_SIP\_PROXY\_RECV\_TERM\_AUTH\_REQUIRED,  
 6 = GT\_SIP\_PROXY\_RECV\_TERM\_RING,  
 7 = GT\_SIP\_PROXY\_FRWD\_TERM\_RING\_TO\_ORIG,  
 8 = GT\_SIP\_PROXY\_RECV\_TERM\_FINAL\_RESP,  
 9 = GT\_SIP\_PROXY\_FRWD\_TERM\_FINAL\_RESP\_TO\_ORIG,  
 10 = GT\_SIP\_PROXY\_RECV\_ORIG\_ACK,  
 11 = GT\_SIP\_PROXY\_FRWD\_ORIG\_ACK\_TO\_TERM,  
 12 = GT\_SIP\_PROXY\_SESSION\_CONNECTED,  
 13 = GT\_SIP\_PROXY\_RECV\_REINVITE,  
 14 = GT\_SIP\_PROXY\_FRWD\_REINVITE,  
 15 = GT\_SIP\_PROXY\_RECV\_REINVITE\_RESP,  
 16 = GT\_SIP\_PROXY\_FRWD\_REINVITE\_RESP,  
 17 = GT\_SIP\_PROXY\_RECV\_REINVITE\_ACK,  
 18 = GT\_SIP\_PROXY\_FRWD\_REINVITE\_ACK,  
 19 = GT\_SIP\_PROXY\_RECV\_REQUEST,  
 20 = GT\_SIP\_PROXY\_FRWD\_REQUEST,  
 21 = GT\_SIP\_PROXY\_RECV\_RESP,  
 22 = GT\_SIP\_PROXY\_FRWD\_RESP,

```

23 = GT_SIP_PROXY_RECV_BYE,
24 = GT_SIP_PROXY_FRWD_BYE,
25 = GT_SIP_PROXY_RECV_BYE_OK,
26 = GT_SIP_PROXY_FRWD_BYE_OK,
27 = GT_SIP_PROXY_RECV_ORIG_CANCEL,
28 = GT_SIP_PROXY_SENT_ORIG_CANCEL_RESP_TO_ORIG,
29 = GT_SIP_PROXY_FRWD_ORIG_CANCEL_TO_TERM,
30 = GT_SIP_PROXY_RECV_TERM_RESP_TO_CANCEL,
31 = GT_SIP_PROXY_SENT_TERM_ACK_OF_CANCEL,
32 = GT_SIP_PROXY_TRANS_VMB_SENT_CANCEL_TO_TERM,
33 = GT_SIP_PROXY_TRANS_VMB_GET_CANCEL_RESP_FROM_TERM,
34 =
GT_SIP_PROXY_TRANS_VMB_WAITING_FOR_INVITING_VMB_ADDRESS,
35 = GT_SIP_PROXY_WAITING_FOR_TERM_ADDR_FROM_APP,
36 = GT_SIP_PROXY_CONNECTION_STEP1,
37 = GT_SIP_PROXY_CONNECTION_DONE,

```

FromID, ToID, RequestID, ViaAddr: Call's Info

RealContactAddr: The real final address this call is trying to reach.

Result\_Code: 0 means no error.

Result\_Txt: Result in text.

**Return:**

null

**Sample code:**

## 4.4.5 ProxySetNewCallSessionAddr

**Description:** Set a new call session's destination address

**Format:**

C++: GT\_BOOL ProxySetNewCallSessionAddr(unsigned int pid, unsigned int sid, const char\* sAddr);

.NET: bool ProxySetNewCallSessionAddr(uint pid, uint sid, string sAddr);

OCX: long ProxySetNewCallSessionAddr(long ProxyID, long SessionID, BSTR DestSIPAddr);

DLL: void GTAPI\_ProxySetNewCallSessionAddr(unsigned int pid, unsigned int sid, const char\* sAddr);

**Parameters:**

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

sid: Call Session ID.

sAddr: The destination address. It can be one the following values:

"**channel**" = to channel. Default value. Then one of the channels will have On\_RecvOffered event triggered.

"**unauthorized**" = need credit to process. You should check the fromID, to see if it is a call from proxy site's user(or extension of PBX). If it does, and Credit is false, then "unauthorized" string is returned to ask authorization to call out.

"**sip:123@abc.com**" = with right credit or don't need credit, processed by proxy site and transfered to this address

"|" can be used to separate the credit info. For example:

"To<sip:123@abc.com>|From<sip:456@abc.com>|User<1234>|Password<4567>"

Otherwise, if none of above matched, refer to user's contact info.

### Return:

null

### Sample code:

## 4.4.6 On\_ProxyUserSubscribed

**Description:** Added an event for proxy user subscribed on the system

### Format:

C++: void On\_ProxyUserSubscribed(unsigned int pid, const char\* fromid, const char\* toid, const char\* suri, const char\* via, const char\* callid, const char\* saddr, unsigned short nport);

.NET: void On\_ProxyUserSubscribed(uint pid, string fromid, string toid, string suri, string via, string callid, string saddr, ushort nport);

OCX: void On\_ProxyUserSubscribed(uint pid, BSTR fromid, BSTR toid, BSTR suri, BSTR via, BSTR callid, BSTR saddr, ushort nport);

DLL: void On\_ProxyUserSubscribed(unsigned int pid, const char\* fromid, const char\* toid, const char\* suri, const char\* via, const char\* callid, const char\* saddr, unsigned short nport);

### Parameters:

pid: Proxy Site ID. Usually it is 0 if you only set one proxy site.

fromid: SIP From

toid: SIP To

suri: SIP Request URI

via: SIP Via header.

callid: call id  
saddr: SIP message source ip address  
nport: SIP message source ip port

**Return:**  
null

**Sample code:**

## 5 SDK Configuration Tags

There are lots of tags to set SDK configuration. Some of them are static(must be set before StartServer is invoked). Some of them are dynamical(can be changed during running). Please refer to webpage: <http://www.pcbest.net/sipsdkcfg.php>.

### **"gtsrv.sip.server.model"**

Server application please set it to "1".  
Softphone application please set it to "0".

### **"gtsrv.sip.protocol"**

Set SIP to run on UDP or TCP or both.  
"1" = UDP, default "2" = TCP "3" = Both UDP and TCP

### **"gtsrv.sip.ip.address"**

The local IP address that SIP application is listening on. Defaultly it is "", means it will work on all IP addresses. If you only want the SIP application running in one specific IP address, please give it explicitly here.

### **"gtsrv.sip.ip.port"**

The local port number of SIP. It can be any value from 1024 to 65535. Statndard port is 5060. It is static, so set it before calling StartServer.

### **"gtsrv.sip.rtpstartrange" and "gtsrv.sip.rtpendrange"**

The local RTP port range. It can be anything from 10000 to 65535.

### **"gtsrv.sip.rtp.port.space"**

Default it is 2. Setting the RTP space of each channel. The best value is 10-20.

### **"gtsrv.net.port"**

The internal exchange port. Set it a value between 8000 and 10000.

**"gtsrv.log.level"**

0 = no log  
1 = error  
2 = alert + error  
3 = debug + alert + error  
4 = info + debug + alert + error  
5 = all

**"gtsrv.log.filename"**

Log file full path.

**"gtsrv.log.file.number"**

How many log files you need to have. Default it is 10.

**"gtsrv.log.size"**

Log file size, in bytes.

**"gtsrv.sip.use.nat.addr"**

Set it to "1" if the SIP proxy server is in local network. It will help SDK core to determine what IP address to use for SIP messages. You don't have to set it. SDK core also can smartly decide what IP address to use.

**"gtsrv.sip.stun.server"**

The stun server to be used to discover global IP address.

**"gtsrv.sip.prefered.codec"**

The order and prefered codecs.

18 - G.729a/b  
102 - Speex.  
104 - iLBC 30ms  
103 - iLBC 20ms  
3 - GSM  
98 - G.726-32  
0 - G.711 mulaw  
8 - G.711 alaw

Sample that only have mulaw and alaw: "0,8".

**"gtsrv.licence.key"**

License key like "ABCDE-FGHIJ-xxxxx-.....",

**"gtsrv.lic.usb.key.driver"**

If use USB key instead, set the USB key driver like "G:".

**"gtsrv.lic.file.dir"**

a tag for setting license file directory.

**"gtsrv.sip.channum.per.span"** (can be **1-30**)  
**"gtsrv.sip.spannum.per.board"** (can be **1-16**)  
**"gtsrv.sip.boardnum.per.server"** (can be **1-20**)  
Set 2.3 above

**"gtsrv.sip.callcontrol.auto.answercall"**  
Set it to 1 to answer incoming calls automatically. Defaultly it is 0.

**"gtsrv.sip.callcontrol.auto.transfercall"**  
If auto-transferring is enabled.

**"gtsrv.sip.dtmf.method"**  
0 = inband  
1 = SIP INFO  
2 = RTP(RFC 2833)  
3 = Auto(Inband or RTP), Default value

**"gtsrv.sip.reg.client.num"**  
How many SIP accounts the phone will register. It is "1" usually.

**"gtsrv.sip.reg1.displayname"**  
The display name of first account. Sample: "Bob Wall"

**"gtsrv.sip.reg1.username"**  
The user name of first account. Sample: "12345678"

**"gtsrv.sip.reg1.domain"**  
The domain of first account. Sample: "pcbest.net"

**"gtsrv.sip.reg1.proxy"**  
The proxy of first account. It is same as domain usually. Sample: "pcbest.net"

**"gtsrv.sip.reg1.authorization"**  
The authorization code of first account. It is same as username usually. Sample:  
"12345678"

**"gtsrv.sip.reg1.password"**  
The password of first account.

**"gtsrv.sip.reg1.expire"**  
The expire time to be registered on the SIP server, in seconds. Defaultly it is 3600, which is one hour.

**"gtsrv.sip.reg1.register"**



If this tag is set to "0", this account will not be registered on the SIP server to take incoming calls, but the phone can still use this account to make outbound calls. Defaultly it is "1".

### **"gtsrv.sip.dxsound.device"**

The key word of your sound capture and playback device. You don't need to set it usually if the default windows playback and capture device is the sound hardware your want to use with ActiveX phone.

### **"gtsrv.sip.dxsound.device.playback"**

The key word of your sound device for playing sound. If you want to use different sound device to playback, then set this value.

### **"gtsrv.sip.dxsound.device.capture"**

The key word of your sound device for recording sound. If you want to use different sound device to record audio, then set this value.

### **"gtphone.audio.record.enabled"**

If enable audio-record. This tag can be changed dynamically when application running, so softphone can dynamically open recording according to users' choice.

1 = enable audio-record.

0 = no audio-record

### **"gtphone.audio.record.rootdir"**

Root folder for audio-record.

### **"gtsrv.dx.play.agc.level" and "gtsrv.dx.capture.agc.level"**

These two tags are for DirectX AGC(Automatic Gain Control) process. the agc level is the percent of max volume for the sound buffer. The valid range is 0.5f to 1.0f, but the recommended range is 0.8f to 0.95f. At 1.0f, some clipping might be experienced. Sample: `CFG_SetValue("gtsrv.dx.capture.agc.level", "0.85")`

### **"gtsrv.sip.vad.enabled"**

If enable VAD(Voice Activity Detection). "1" = enable. "0" = no

### **"gtsrv.sip.conference.room"**

Conference room number. Set how many conference room to open.

### **"gtsrv.sip.on.in.vad" and "gtsrv.sip.on.out.vad"**

For VAD(Voice Activity Detection). Set it to 1 to enable. Defaultly it is disabled(0). On\_VoiceActivityDetected event will be triggered if it is on.

### **"gtsrv.sip.callcontrol.auto.acceptcall"**

Defaultly it is 1, means accpet call automatically and send SIP TRY response once the call is in. If you set it to 0, then you have to use Send\_Accept later to accept the call.

**"gtsrv.sip.callcontrol.auto.ringcall"**

Defaultly it is 1, means send SIP 180 RING response once the call is in. If you set it to 0, then you have to use Send\_Ring later to ring the call.

**"gtsrv.sip.on.rtp.packet"**

Defaultly it is 0, means no accessing to rtp package.

If it is 1(MULAW), 2(ALAW), 3(PCM), it means On\_RecvRTPPacket and On\_SentRTPPacket events will be triggered to access RTP package.

If it is -1, it means On\_RTPRawPacket event will be triggered to access RTP raw package.

**"gtsrv.sip.on.dx.audio"**

Defaultly it is 0, means no accessing to directx audio stream. If it is 1(MULAW), 2(ALAW), 3(PCM), it means On\_CaptureDXAudio and On\_RenderDXAudio events will be triggered to access RTP package.

**"gtsrv.sip.proxy.sites.num"**

How many proxy sites. Default it is 0. For PBX application, it is 1 usually.

**"gtsrv.sip.proxy.site1.domain"**

The domains to handle for the SIP proxy. You can set multiple domains by saperating them by ";".

**"gtsrv.sip.proxy.site1.recordroute"**

Default it is 0, means no.

**"gtsrv.sip.proxy.site1.udp.relay"**

Default it is 0, means no.

**"gtsrv.sip.get.totag.from.ring.for.cancel"**

SIP server/gateway specific tag. In the case, you can't cancel a call, please set it to 0.

Defaultly it is 1, means it will learn the totag from 180 RINGING message and use it for CANCEL.

//JB buffer and directX buffer settings

**"gtsrv.sip.jb.min.delay"**

Default 0. You can set it to 2 -5 if improve audio smoonth.

**"gtsrv.sip.jb.max.delay"**

Default 5. Set it from 5 -25.

**"gtsrv.sip.jb.adaptive"**

Default 0. Set it to 1 for adaptive.

**"gtsrv.sip.jb.get.all.data"**

Default 1. Set it to 0 for not getting all data.

**"gtsrv.sip.dx.playback.delay"**

Default 2, means 2 x 20ms delay in DirectX buffer.

**"gtsrv.sip.use.prack"**

Default it is 0. Not using PRACK. You can set it to 1 to enable PRACK feature.

**"gtsrv.sip.max.dlg.duration"**

Defaultly it is 21600 seconds(6 hours).

You can set it to 0, for unlimited.

**"gtsrv.sip.enable.dns.srv"**

Default it is 0. Set it to 1 if you want to use DNS.

**"gtsrv.sip.proxy.site1.check.user.status"**

default is false(0). Set it to 1 to enable it.

**"gtsrv.sip.keep.alive.msg"**

Default it is 1, sending keep alive message. Used for making a whole on router so outside messages can go in. Set it to 0 if not needed, especially within NAT.

**"gtsrv.sip.get.remote.contact.from.ring"**

Some GWs or SIP servers only recognize the SIP URI in Ring or SessionProgress they give when cancelling a call.

Other only recognize the initial SIP URI in SIP INVITE when cancelling a call even they give a different SIP contact address in ring, or session progress.

In order to be compatible to both behaviors, this option is added.

Default it is FALSE(0). You can set it to TRUE(1) if you are having problem to cancel an outbound call.

**"gtsrv.sip.callcontrol.handle.3xx"**

To control if the SDK should automatically deal with SIP 3xx message.

**"gtsrv.sip.dyn.check.remote.rtp"**

Defaultly it is 1 = enabled. Set it to 0 means disable checking

**"gtsrv.sip.public.ip.in.rtp"**

a tag to set public ip address instead of using STUN

sometimes if your machine is on DMZ, then you can set this tag.

Set it to "" = not enabled

Set it to "0" will always let SDK to use public ip address in SIP SDP.

See it to "209.223.21.33" as using this public ip address

**"gtsrv.sip.dtmf.key.duration"**

setting for inband audio when playing DTMF string

this.environment.CFG\_SetValue("gtsrv.sip.dtmf.key.duration", "100");

this.environment.Send\_PlayDTMFStr(connection.Channel, t.Tone);

**"gtsrv.sip.min.inband.audio.dtmf.dur"**

setting the minimal inband DTMF tone length for DTMF detection.

The default value for it is 120ms. You can set it to 40ms plus, but remeber, the lower value, the more sensitive it could be, so lower value might cause problem.