PCBest Networks SIP SDK API Reference

(For SDK V2.05f)

Copyright 2011 PC Best Networks Inc.

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 31
    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 40
    3.2.12 Send_SessionProgress 41
    3.2.13 Send_WaitForCall (new function from 2.05f) 41
    3.2.14 On_RecvConnected 42
    3.2.15 On_RecvOffered 43
    3.2.16 On_RecvDialing 43
    3.2.17 On_RecvRinging 44
    3.2.18 On_RecvIdle (New Format) 45
    3.2.19 On_RecvSessionProgress 45
    3.2.20 On_RecvHolding 46
    3.2.21 On_RecvTransfering 47
    3.2.22 On_RecvNoAudio 47
    3.2.23 GetChanCallID 48
    3.2.24 GetChanCallFromTag 49
    3.2.25 GetChanCallToTag 49
  3.3 Channel Status Functions 50
    3.3.1 Send_GetChanStatus 50
    3.3.2 On_RecvStatus 51
    3.3.3 Send_SetChanStatus 51
    3.3.4 SetChanDir (New function since 2.05f) 52

3.4 SIP Accounts Status Functions 53
  3.4.1 Set SIP Accounts 53
  3.4.2 SIP Accounts Register Event(On_RecvRegStatus) 54
  3.4.3 Send_GetRegStatus 55
3.5 Dynamical SIP Account Management (New Feature of SDK v1.71f) 55
  3.5.1 SIPAccount_Add 55
  3.5.2 SIPAccount_Remove 56
  3.5.3 SIPAccount_Count 57
  3.5.4 SIPAccount_Index 57
  3.5.5 SIPAccount_Handle 57
  3.5.6 SIPAccount_Get 58
  3.5.7 SIPAccount_Enable 59
3.6 DTMF Functions and Events 59
  3.6.1 Send_PlayDTMFStr 59
  3.6.2 Send_EnableDTMF 60
  3.6.3 Send_DisableDTMF 60
  3.6.4 On_RecvDTMFDone 61
  3.6.5 On_RecvDTMFKeyDown 61
  3.6.6 On_RecvDTMFKeyUp 62
3.7 CT_BUS Functions 63
  3.7.1 Send_DuplexConnect 63
  3.7.2 Send_DuplexDisconnect 64
  3.7.3 Send_HalfConnect 64
  3.7.4 Send_HalfDisconnect 65
  3.7.5 Send_RTPDuplexConnect 65
  3.7.6 Send_RTPDuplexDisconnect 66
3.8 Audio Control Functions and Events 66
  3.8.1 Send_PlayAudio 67
  3.8.2 Send_RecordAudio 67
  3.8.3 Send_RecordAudio2 68
  3.8.4 Send_AddAudio 69
  3.8.5 Send_ClearAudio 70
  3.8.6 Send_StopAudio 70
  3.8.7 Send_StopAudioEx 71
  3.8.8 Send_SetAudioFormat 71
  3.8.9 Send_GetAudioStatus 73
  3.8.10 Send_StartDXAudio 73
  3.8.11 Send_StopDXAudio 74
  3.8.12 Send_ResetDXAudio 75
  3.8.13 Send_SetDXAudioVolume 75
  3.8.14 On_RecvAudioPlayDone 76
  3.8.15 On_RecvAudioRecordDone 76
  3.8.16 On_RecvAudioStatus 77
  3.8.17 On_RecvDXAudioStatus 78
3.9 MOH(Music On Hold) Functions and Events 79
  3.9.1 Send_StartMusicOnHold 79
  3.9.2 Send_StopMusicOnHold 80
  3.9.3 Softphone “Hold” implementation 80
3.10 Conference Functions and Events 81
  3.10.1 Send_StartConference 81
  3.10.2 Send_StopConference 81
  3.10.3 Send_SetChanInConference 82
  3.10.4 CreateConf 83

3.10.5 DestroyConf 84
3.10.6 SetChanInConf 84
3.10.7 GetConfIndex 85
3.10.8 SetChanConfMask 85
3.11 Conference Audio Functions and Events 86
3.11.1 Send_ConfPlayAudio 86
3.11.2 Send_ConfRecordAudio 87
3.11.3 Send_ConfRecordAudio2 88
3.11.4 Send_ConfAddAudio 88
3.11.5 Send_ConfClearAudio 88
3.11.6 Send_ConfStopAudio 89
3.11.7 Send_ConfStopAudioEx 90
3.11.8 Send_GetConfAudioStatus 90
3.11.9 Send_ConfSetAudioFormat 91
3.11.10 On_RecvConfAudioStatus 92
3.11.11 On_RecvConfAudioPlayDone 93
3.11.12 On_RecvConfAudioRecordDone 94
3.12 Tone Detection Functions and Events 95
3.12.1 Send_AddTone 95
3.12.2 Send_ClearToneList 95
3.12.3 Send_StartToneDetection 96
3.12.4 Send_StopToneDetection 96
3.12.5 On_RecvToneDetected 97
3.13 VAD Functions and Events 97
3.13.1 On_VoiceActivityDetected 97
3.13.2 Send_EnableVAD 98
3.13.3 Send_DisableVAD 99
3.14 Instant Message Fonctions and Events 99
3.14.1 Send_MessageText 99
3.14.2 On_RecvMessageTextDelivered 100
3.14.3 On_RecvMessageText 100
3.15 Support for SIP/Presence 101
3.15.1 RecvSubscribeStatus 101
3.15.2 RecvNotifyPresence 102
3.16 RTP and DirecX Audio Data Events 102
3.16.1 On_RecvRTPPacket 103
3.16.2 On_SentRTPPacket 104
3.16.3 On_CaptureDXAudio 105
3.16.4 On_RenderDXAudio 106
3.17 Softphone Assistant Functions 107
3.17.1 PlayNumTone 107
3.17.2 PlayLocalRingSound 107
3.17.3 PlayRemoteRingSound 108
3.17.4 PlayBusySound 108
3.17.5 StopSound 109
3.17.6 SetMicVolume 110
3.17.7 GetMicVolume 110
3.17.8 SetSpeakerVolume 111
3.17.9 GetSpeakerVolume 111
3.17.10 SetSpeakerMuteStatus 112
3.17.11 GetSpeakerMuteStatus 112
3.17.12 SetMicMuteStatus 113
3.17.13 GetMicMuteStatus 113

3.17.14 GetSoundDeviceCount deprecated 114
3.17.15 GetSoundDeviceName deprecated 114
3.17.16 GetRenderDeviceCount 115
3.17.17 GetRenderDeviceName 115
3.17.18 IsRenderDevicePrimary 116
3.17.19 GetCaptureDeviceCount 116
3.17.20 GetCaptureDeviceName 117
3.17.21 IsCaptureDevicePrimary 117
3.18 Network Functions 118
3.18.1 GetLocalIPCount 118
3.18.2 GetLocalIP 118
3.18.3 GetLocalNetworkIPAddress deprecated 119
3.18.4 GetSIPAddressInfo 119
3.18.5 GetDetectedNATType 120
3.18.6 GetMappedPublicSIPIPAddress 121
3.18.7 GetMappedPublicSIPIPPort 121
3.18.8 GetLocalSIPPort 122
3.18.9 GetLocalRTPPort 122
3.18.10 GetPeerSIPIPAddress 122
3.18.11 GetPeerSIPIPPort 123
3.18.12 GetPeerRTPIPAddress 124
3.18.13 GetPeerRTPIPPort 124
3.18.14 GetPeerSIPContactAddress 125
3.19 Error Event 125
3.19.1 On_RecvError 125
3.20 Timer functions 127
3.20.1 StartTimer 127
3.20.2 StopTimer 127
3.20.3 OnTimer 128
3.21 SDK License Functions 128
3.21.1 SetAppName 129
3.21.2 GetLicTo 129
3.21.3 Tags for license 130
3.22 Human voice or Answering Machine Detection 131
3.22.1 On_DetectHumanVoiceDone 131
3.23 Other functions 131
3.23.1 GetChanAudioCodec 131
3.23.2 GetChanLastMsgCode 132
3.23.3 GetChanLastMsgText 133
3.23.4 SetChanUserData 134
3.23.5 GetChanUserData 135
3.23.6 SetMainWnd 135
3.23.7 GetTotalChannelNumber 136
3.23.8 GetChanAudioRecordFileName 136
3.23.9 SetChanCallExtraSIPHeader 137
3.23.10 SetChanAudioLevel 138
4 Proxy API 138
4.1 Initialize Proxy Site 138
4.1.1 InitProxySites 139
4.2 Free Proxy Site 140
4.2.1 FreeProxySites 140
4.3 Add, Delete, Change or Disable Proxy Site User 140
4.3.1 ProxySetUserInfo 140

4.3.2 ProxyDisableAllUsers 142
4.4 Events 142
4.4.1 On_ProxyUserRegistered 142
4.4.2 On_ProxyNewCallSession 144
4.4.3 On_ProxyCallSessionStatus 145
4.4.4 On_ProxyCallTransaction 146
4.4.5 ProxySetNewCallSessionAddr 148
4.4.6 On_ProxyUserSubscribed 149
5 SDK Configuration Tags 150

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
Toll Free(USA & Canada): 1-866-807-0528
Local and International: 1-613-239-4278

1.1 key features and specifications

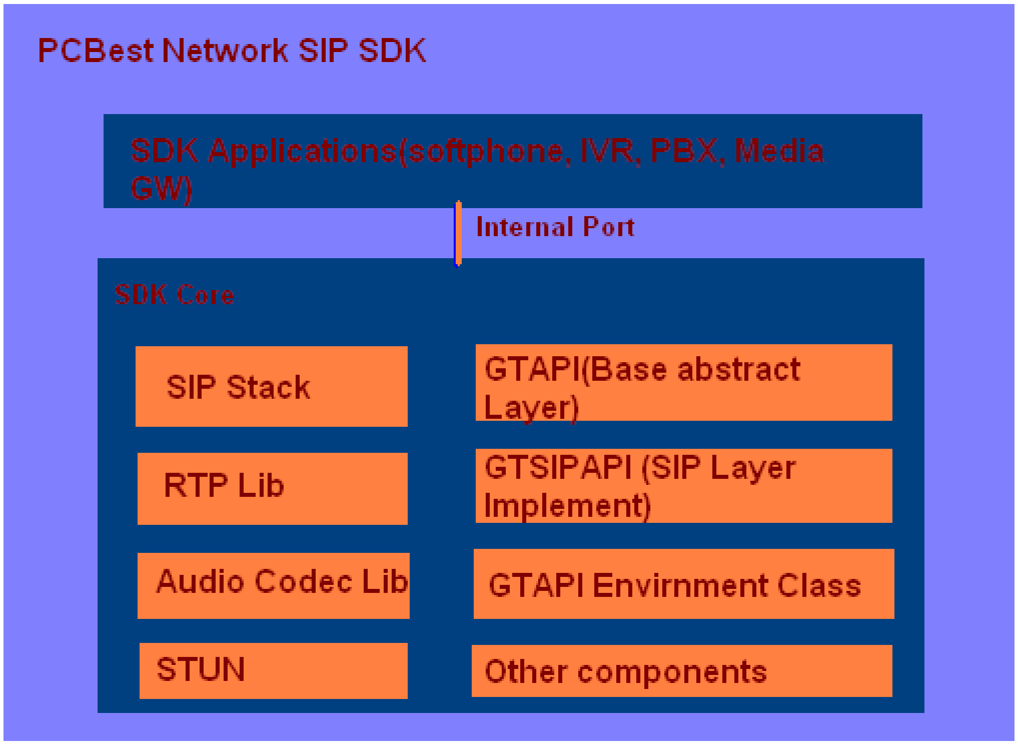
SIP stack	PCBest Network SIP Stack(completely own and developed by PCBest. We can decode any additional features in SIP message for your SIP Project, or fix any problems that may exist in the SIP core.)	Very stable and compact size
Compatible SIP Servers, Proxy and PBX	Full compatible with Open SER, 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+	
Programming interfaces	C++ head files and lib .NET assembly(managed interfaces) ActiveX control Standard DLL Interface	

Supported development tools	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	
Audio Record (Dynamically turn on in a live call)	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,	

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)	
RTP Package Access	Support access incoming and outgoing RTP audio stream directly. And support **change RTP audio stream to integrate TTS and ASR engine**	*****Very powerful feature.
DirectX Audio Stream Access	Yes. Can Access and change the DirectX audio on the middle way on both play and record direction	*****Very powerful feature.
Support dynamically change sound devices during a live call	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	
Conference	YES	*****

Inbound Voice Activity Detection(VAD)	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	
Tone Detection	YES	Good for fax detection and ring tone detection

1.2 SDK Component Architecture


PCBest Network SIP SDK
SDK Applications(softphone, IVR, PBX, Media GW)
Internal Port
SDK Core
SIP Stack
RTP Lib
Audio Codec Lib
STUN
GTAPI(Base abstract Layer)
GTSIPAPI (SIP Layer Implement)
GTAPI Envirnment Class
Other components

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\GTF2Test
A C++ sample to show how to access SDK by using standard DLL interfaces. It implements a simple SIP phone.

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

SDK\Samples\OCX\BCB6
Borland C++ version of simple SIP phone.

SDK\Samples\OCX\Delphi6
Delphi version of simple SIP phone

SDK\Samples\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.

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.

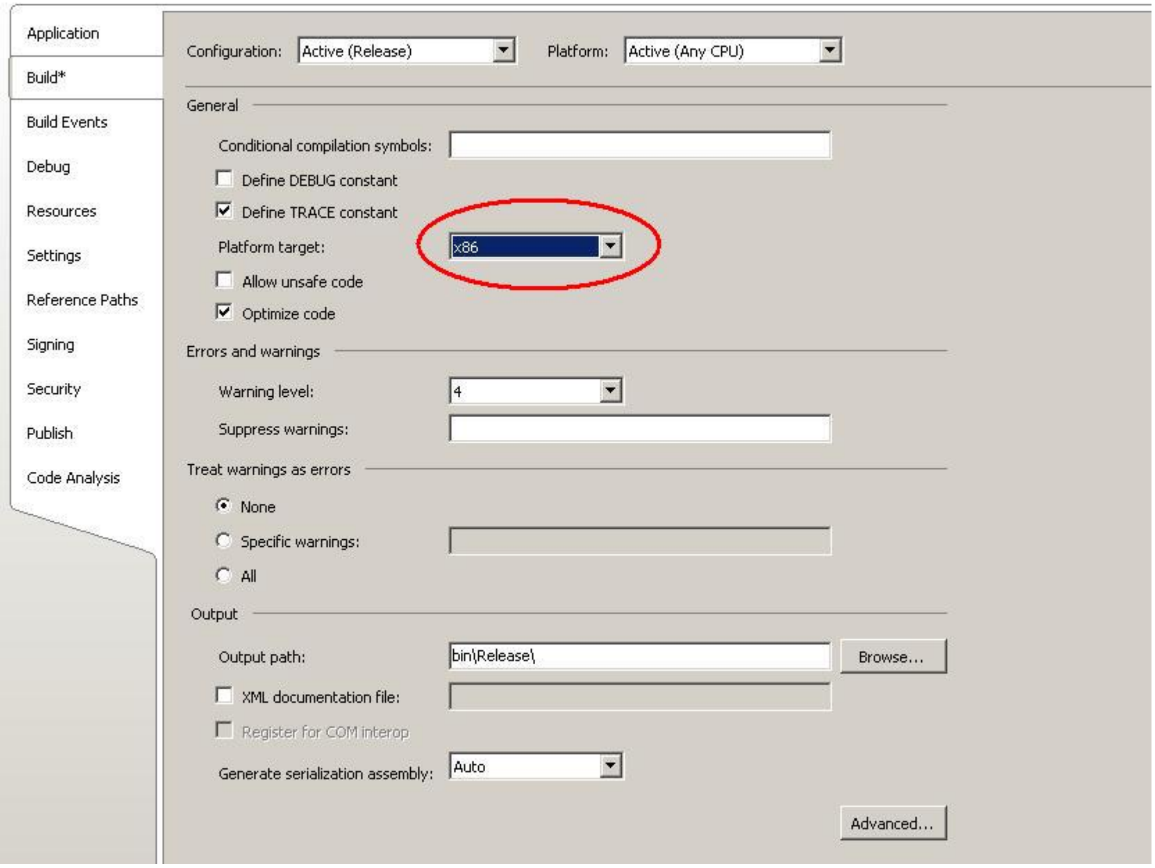
For Standard DLL Interface Developers

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:


Application
Build*
Build Events
Debug
Resources
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Browse...
Advanced...

Configuration: Active (Release)
Platform: Active (Any CPU)

General

Conditional compilation symbols:
☐ Define DEBUG constant
☑ Define TRACE constant
Platform target: x86
☐ Allow unsafe code
☑ Optimize code

Errors and warnings

Warning level: 4
Suppress warnings:

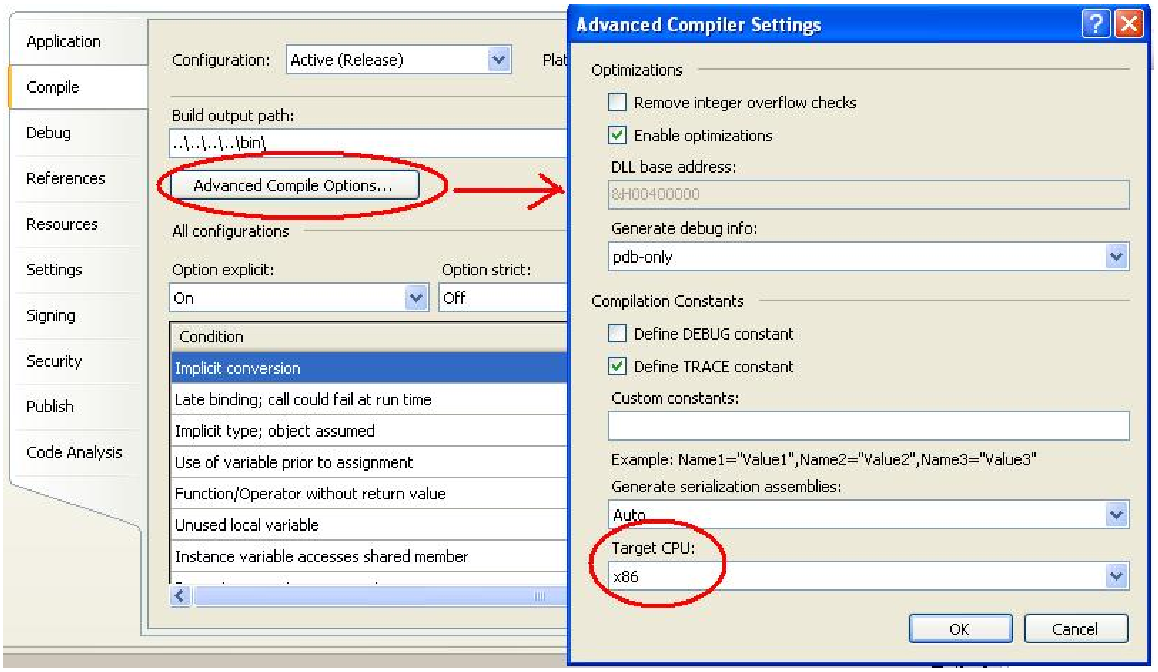
Treat warnings as errors

● None
○ Specific warnings:
○ All

Output

Output path: bin\Release\
☐ XML documentation file:
☐ Register for COM interop
Generate serialization assembly: Auto

VB.NET project should have settings like this:


Application
Compile
Debug
References
Resources
Settings
Signing
Security
Publish
Code Analysis
Advanced Compile Options...
Advanced Compiler Settings
OK
Cancel

Configuration: Active (Release)
Build output path: ..\..\..\..\bin\

All configurations

Option explicit: On
Option strict: Off

Condition
Implicit conversion
Late binding; call could fail at run time
Implicit type; object assumed
Use of variable prior to assignment
Function/Operator without return value
Unused local variable
Instance variable accesses shared member

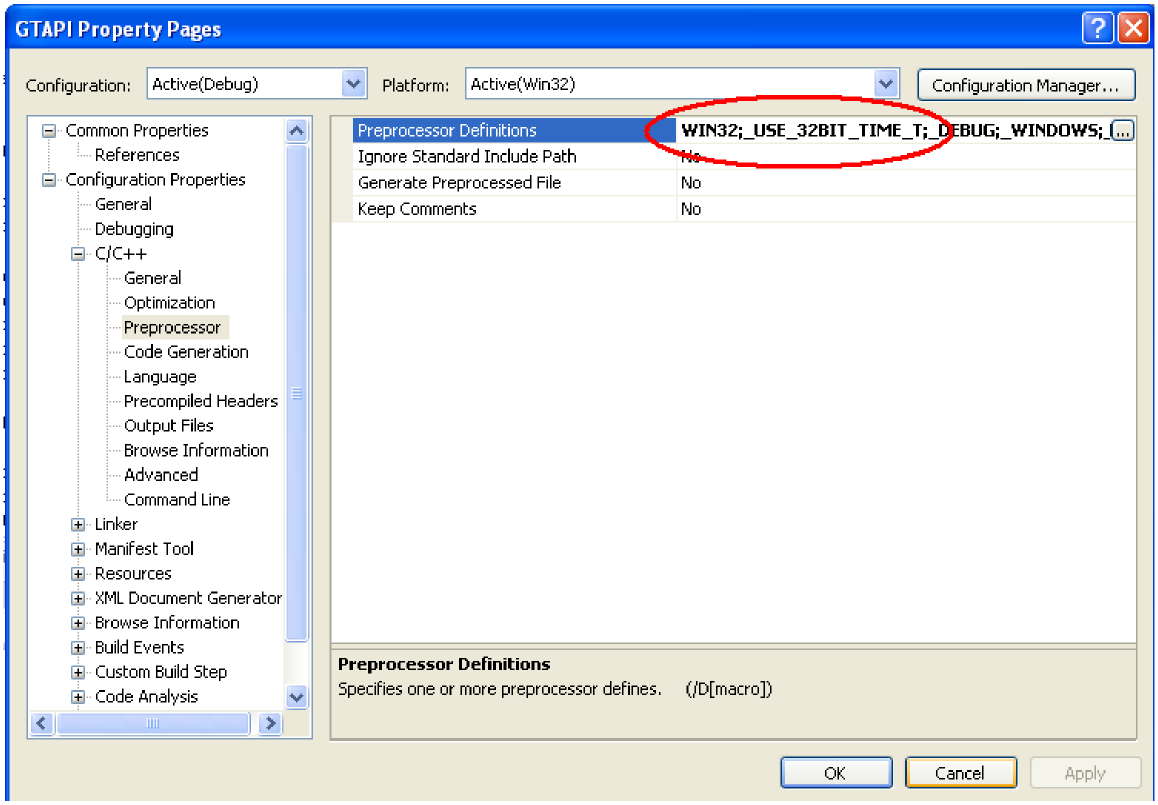
Optimizations

☐ Remove integer overflow checks
☑ Enable optimizations
DLL base address: &H00400000
Generate debug info: pdb-only

Compilation Constants

☐ Define DEBUG constant
☑ Define TRACE constant
Custom constants:
Example: Name1="Value1",Name2="Value2",Name3="Value3"
Generate serialization assemblies: Auto
Target CPU: x86

C++ projects should define WIN32 and _USE_32BIT_TIME_T:


GTAPI Property Pages
Configuration Manager...
Common Properties
References
Configuration Properties
General
Debugging
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
Manifest Tool
Resources
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
OK
Cancel
Apply

Configuration: Active(Debug)
Platform: Active(Win32)

Preprocessor Definitions	WIN32;_USE_32BIT_TIME_T;_DEBUG;_WINDOWS;
Ignore Standard Include Path	No
Generate Preprocessed File	No
Keep Comments	No

Preprocessor Definitions
Specifies one or more preprocessor defines. (/D[macro])

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 inited
    Else
            //inited
    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 inited
  }
}

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 inited
         }
      }

      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	(cab 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.per.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_RecvTransfering
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. The SDK will issue On_RecvIdle event when the call disconnected successfully.

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,
unpdate 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 foeget 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 people prefer to allow inbound call only after calling a funciton 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 "gtsrv.wait.for.call.mode" to 1 before startServer, then call this function after channel is IDLE. Note: if you use this mode, and don't call Send_WaitForCall on the channel, this channel will not be able to accept incoming call.

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 inbound calls.

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 route. 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_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");
}
```

3.2.17 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();
}
```

3.2.18 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.");
}
```

3.2.19 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.");
}
```

3.2.20 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.

3.2.21 On_RecvTransfering

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_RecvTransfering(int ch, const char* sAddr, const char* sReplaceCallID, const char* sCallFromTag, const char* sCallToTag)
.NET: void On_RecvTransfering(int ch, string sAddr,string sReplaceCallID, string sCallFromTag, string sCallToTag)
OCX: void OnCallTransfering(int ch, BSTR sAddr, BSTR sReplaceCallID, BSTR sCallFromTag, BSTR sCallToTag)
DLL: void GTAPI_SetCB_On_ RecvTransfering

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_ RecvTransfering (int ch, const char* sAddr, const char* sReplaceCallID,
const char* sCallFromTag, const char* sCallToTag)
{
        Printf("Call is transferring to %s", sAddr);
}
```

3.2.22 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);
 }
```

3.2.23 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);
```

3.2.24 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);
```

3.2.25 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 mannually 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 mannually 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).

dor:

```
// 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(2, 2);
```

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.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", “xxxxxx”);
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);
```

3.5 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)
.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)

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)
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)

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.

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);

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 SIPAccountEnable(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

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 clould 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 logout
Void On_RecvDTMFKeyUp(int ch, unsigned char keyValue, unsigned long ticks)
{
     Printf("%c is released", keyValue);
}
```

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_RTPDuplexDisconnect (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 assistant 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 outdoing** 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_HungUp), 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, itn 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_RecvDXAudioStatu

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_StartMisicOnHold(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_StopMisicOnHold(0);
```

3.9.3 Softphone "Hold" implementation

If you 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. meets.

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 “own timeout”
env.Send_ConfStopAudioEx(0, 1, “own 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, itn 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 Instant Message Fonctions and Events

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

3.14.1 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.”)

3.14.2 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: massage decription.

Return:
null

Sample code:

3.14.3 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:

3.15 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 = pidf(default), 1 = dialog-info

There are two events for subscribe and notify presence.

3.15.1 RecvSubscribeStatus

Description: Event to tell if subcribe 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:

3.15.2 RecvNotifyPresence

Description: Event to tell if subcribe 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/pidf+xml"
content_info: real xml content

Return:
null

Sample code:

3.16 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.

3.16.1 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.
}
```

3.16.2 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
}
```

3.16.3 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);
}
```

3.16.4 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 Softphone Assistant Functions

3.17.1 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’);
```

3.17.2 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();
}
```

3.17.3 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();
}
```

3.17.4 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();
}
```

3.17.5 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();
}
```

3.17.6 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);
```

3.17.7 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();
```

3.17.8 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);
```

3.17.9 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();
```

3.17.10 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);

3.17.11 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}

3.17.12 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);
```

3.17.13 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}
```

3.17.14 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();
```

3.17.15 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);
```

3.17.16 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();
```

3.17.17 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);
```

3.17.18 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:

3.17.19 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();
```

3.17.20 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);
```

3.17.21 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:

3.18 Network Functions

3.18.1 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();
```

3.18.2 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();

3.18.3 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();

3.18.4 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 sring
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>”);
```

3.18.5 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 = Independedt Mapping, Address Dependendent Filter
6 = Indepndent Mapping, Port Dependent Filter
8 = Dependent Mapping
10 = Firewalled
12 = Blocked or could not reach STUN server
14 = Unkown 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.”);
}
```

3.18.6 GetMappedPublicSIPIPAddress

Description: This function gets the real public Internet IP address for SIP communication.

Format:
C++: const char* GetMappedPublicSIPIPAddress()
.NET: string GetMappedPublicSIPIPAddress()
OCX: BSTR GetMappedPublicSIPIPAddress()
DLL: const char* GTAPI_GetMappedPublicSIPIPAddress()

Parameters:

Return:
IP address string.

Sample code:
```
//get ip address of sip
env.StartServer();
env.GetMappedPublicSIPIPAddress();
env.GetMappedPublicSIPIPPort();
```

3.18.7 GetMappedPublicSIPIPPort

Description: This function gets the real public Internet IP port for SIP communication.

Format:
C++: unsigned short GetMappedPublicSIPIPPort()
.NET: unsigned short GetMappedPublicSIPIPPort()
OCX: unsigned short GetMappedPublicSIPIPPort()
DLL: unsigned short GTAPI_GetMappedPublicSIPIPPort()

Parameters:

Return:
Port number

Sample code:
```
//get ip address of sip
env.StartServer();
env.GetMappedPublicSIPIPAddress();
env.GetMappedPublicSIPIPPort();
```

3.18.8 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:

3.18.9 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:

3.18.10 GetPeerSIPIPAddress

Description: After the call is connected, this function can be used to get the peer SIP address.

Format:
C++: const char* GetPeerSIPIPAddress(int ch);
.NET: string GetPeerSIPIPAddress(int ch);
OCX: BSTR GetPeerSIPIPAddress(int ch);
DLL: const char* GTAPI_GetPeerSIPIPAddress(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.GetPeerSIPIPAddress(ch));
    }
```

3.18.11 GetPeerSIPIPPort

Description: After the call is connected, this function can be used to get the peer SIP port.

Format:
C++: unsigned short GetPeerSIPIPPort(int ch);
.NET: unsigned short GetPeerSIPIPPort(int ch);
OCX: unsigned short GetPeerSIPIPPort(int ch);
DLL: unsigned short GTAPI_GetPeerSIPIPPort(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.GetPeerSIPIPPort(ch));
    }
```

3.18.12 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));
    }
```

3.18.13 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));
}
```

3.18.14 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));
    }
```

3.19 Error Event

Error event are used to notify application level errors.

3.19.1 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);
}
```

3.20 Timer functions

3.20.1 StartTimer

Description: Start timer on the channel. Later on, an event On_Timer will be triigered 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);
```

3.20.2 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);
```

3.20.3 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);
}
```

3.21 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

3.21.1 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!”);
}
```

3.21.2 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!”);
}
```

3.21.3 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.

3.22 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 setence is less than this threshold, it is considered as a human voice, otherwise it is machine. defaultly it is 2000 (2 seconds)

3.22.1 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:

3.23 Other functions

3.23.1 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));
}
```

3.23.2 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));
}
```

3.23.3 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));
}
```

3.23.4 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:

3.23.5 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:

3.23.6 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();
```

3.23.7 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();
```

3.23.8 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);
}
```

3.23.9 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”);
```

3.23.10 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);

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.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;
```

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* uname, int nattype, const char* src_ip, unsigned short src_port);
.NET: void ProxySetUserInfo(uint pid, string username, string passwd, uint regt, uint regex, string contactaddr, string mapped_contact, string uname, int nattype, string src_ip, ushort src_port);
OCX: void ProxySetUserInfo(uint pid, BSTR username, BSTR passwd, uint regt, uint regex, BSTR contactaddr, BSTR uname, 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 );
```

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.
regex: Registered expiration Seconds.
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

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:

4.4 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* szUAName, 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 szUAName, 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.
szUAName: 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* szUAName, 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 seperate 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 seperate 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.