VoIP SIP SDK

DEVELOPER MANUAL

28.12.2019

Table of Contents

QUICK START

Using SDK

VoIP SIP SDK allows to add audio/video call features to existing application, develop soft-phone with custom business logic and design, automate office tasks via creating auto-dialer and implement many other cases.

SDK abstraction

VoIP SIP SDK implements SIP client functionality and provides high-level abstraction to access it.

connection	Call with remote side. Connection is created when app makes outgoing call or receives incoming. Each connection has own unique `connectionId`, generated by SDK.
line	Container for connections. By default SDK moves each connection to different line. Each line has own **lineId.** It's fixed value in range 1..8;
current line	Selected by app using method: `phone.SetCurrentLine(n)` SDK plays sound, received from connection(s) in current line on local speaker and also send sound from local microphone to connection(s) on current line.
conference	Created by SDK when app moves few connections to same line. In this mode SDK configures internal mixer gains and properly sends sound between local side and all connections (each connection can hear local side and other connections). It's possible to configure mixer gains in way when some connection can only hear, but can't speak. See more: `phone.SetConnectionContribution();`

SDK lifecycle inside app

1. Create 'phone' instance
2. Configure and initialize created 'phone' instance
   SIP/RTP ports, registration credentials, license credentials, log-level,
   audio devices, network interface, codec's list, etc.
   All settings app can serialize/de-serialize using file/db/other ways.
3. Subscribe for events
4. Make/receive calls, send/receive SIP requests.
5. Unsubscribe events receiving, destroy instance.

When is required to modify 'phone' settings app can do this without re-creating existing instance. SDK detects which Config's properties were modified and applies only required changes.

SDK implementation details

SDK is compiled as native dll, written on C++.
Its functionality is available through ActiveX interface or Dll interface (exported functions).

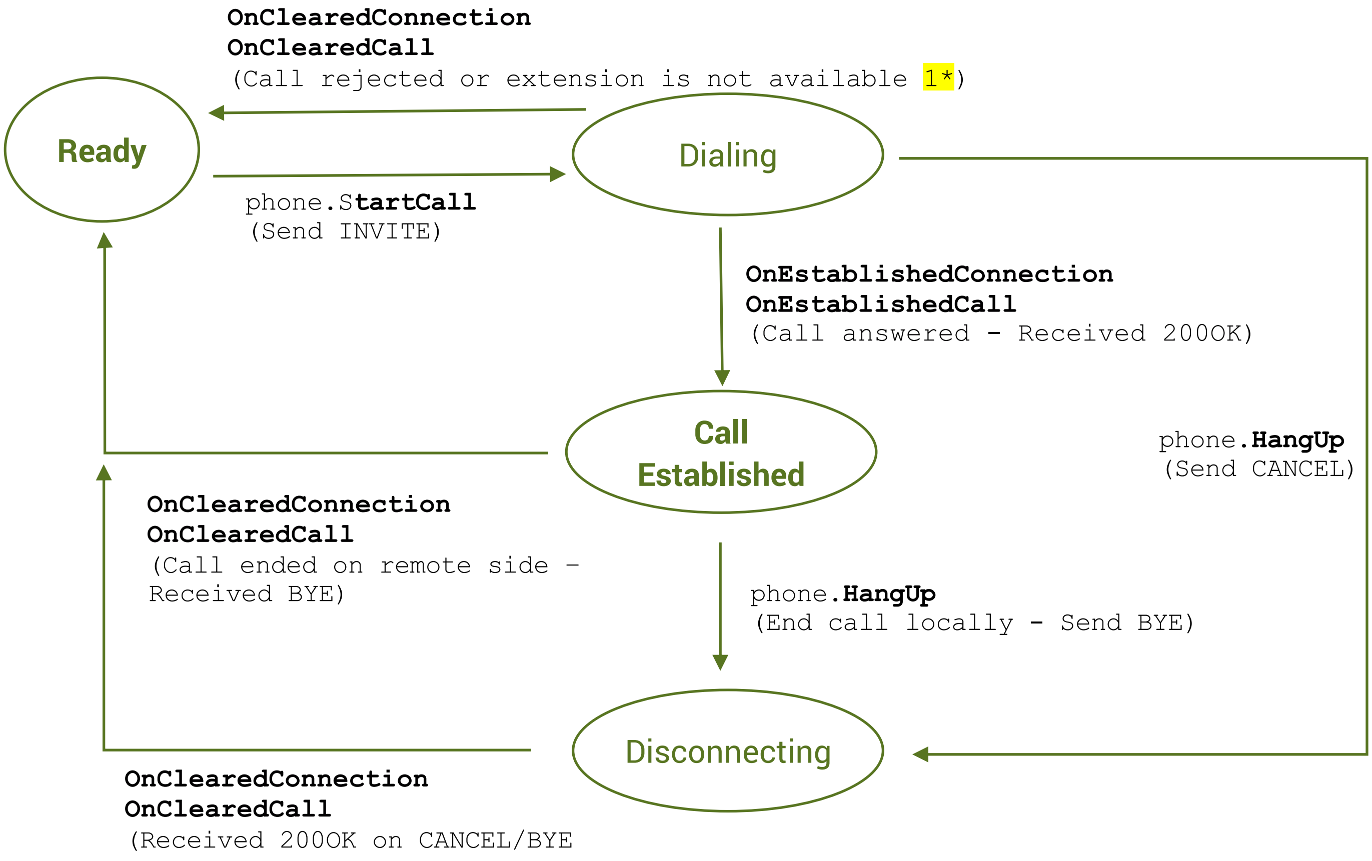
SDKs API is available as set of methods/properties/events joined in 3 interfaces:

- IAbtoPnone – methods related to 'phone' functionality (like `StartCall, AnswerCall, PlayFile` etc);
- IConfig – phone configuration (audio devices, registration accounts, signaling transport, etc);
- ISubscriptions – provides support for presence, voicemail, BLF implementation;

Almost all SDKs methods implemented as async (app just posts command to internal queue and it's handled by background thread) – no need to create own thread inside app.
SDK sends events to app from same thread, where it was created.

Calls handling state diagram

Outgoing call

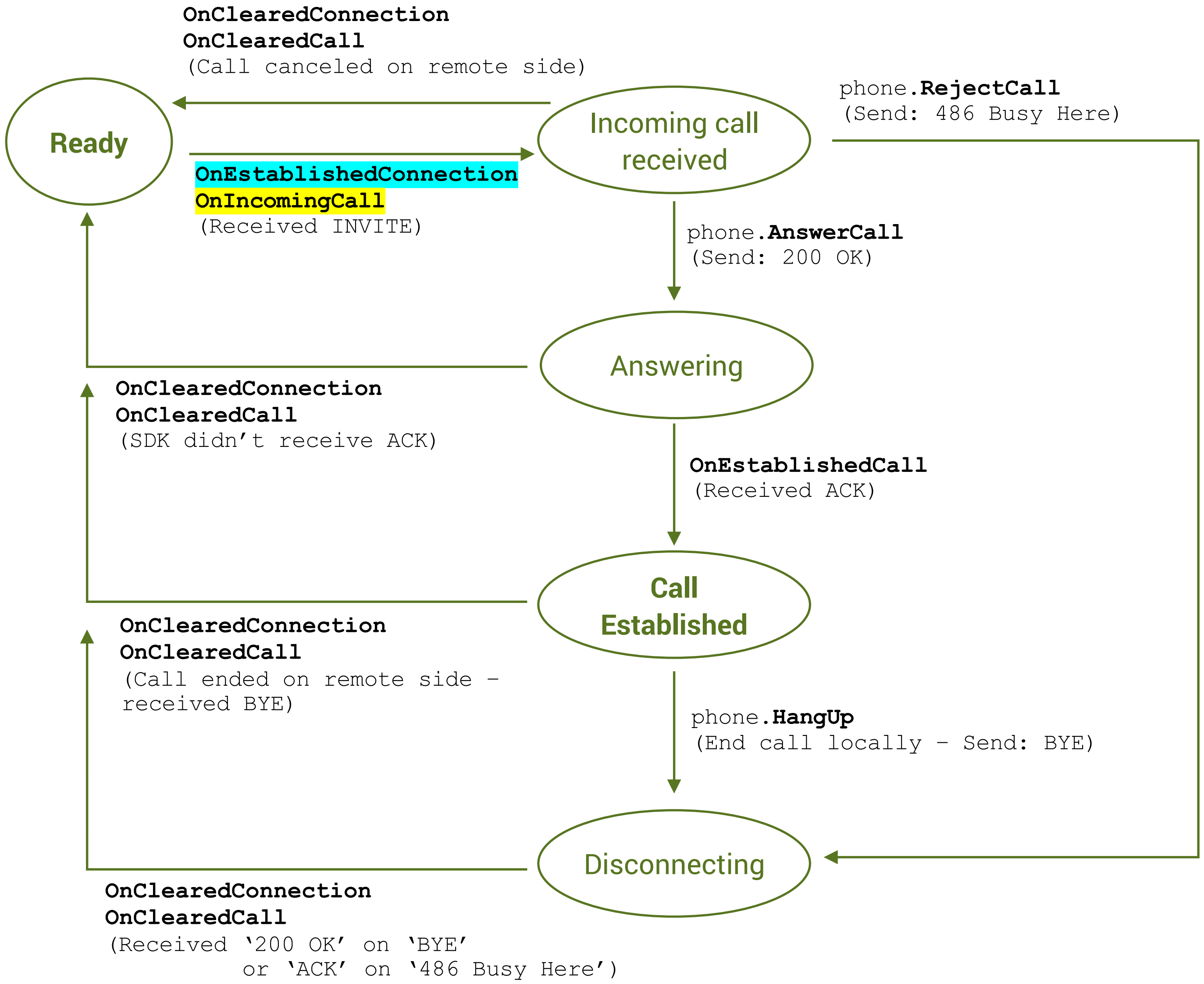


1* - app can modify time, how long SDK is waiting answer on outgoing INVITE request using setting:

```
Phone.config.CallInviteTimeout= 40;//seconds
```

When SDK didn't receive any answer during 'CallInviteTimeout it will end call automatically.

Incoming call



SDK API

Initialize ‘phone’ instance and modify its settings

Name	Description
Initialize	This method initializes created ‘phone’ instance. App has invoke it only once. It makes following actions: • Check license • Check existing output/input audio devices, network interfaces; • Check codec’s list – stops initialization when codecs not found; • Initializes internal components When some error happens during initialization method raises exception (.net app). Method always generates event `OnInitialized`, which allows to detect was initialization successful or which error happens. Syntax: `phone.Initialize();`

InitializeEx	This method does same as `Initialize`, but has one more argument, which allows to configure sending events from different thread.
	
	Syntax:
	`phone.InitializeEx(int sendEventsFromSameThread);`
	
	Notes:
	When `'sendEventsFromSameThread'` is not zero – SDK will send events from internal thread, not from thread where it was created.
	
	Usage example:
	`//Create ‘phone’, assign event handlers`
	`phone = new CAbtoPhone();`
	`phone.OnInitialized += Phone_OnInitialized;`
	
	`//Get ‘config’ interface and set initial settings`
	`CConfig phoneCfg = phone.Config;`
	`phoneCfg.LicenseUserId = "{Trial0…}”;`
	`phoneCfg.LicenseKey = "{SbkqEvE1X…}”;`
	`phoneCfg.LogLevel = LogLevelType).eLogDebug;`
	`phoneCfg.ListenPort = 5080;`
	`phoneCfg.RegDomain = "";`
	`phoneCfg.RegUser = "";`
	`phoneCfg.RegPass = "";`
	`phoneCfg.RegExpire=300;`
	
	`//Apply changes`
	`phone.ApplyConfig();`
	
	`//Initialize ‘phone’`
	`phone.InitializeEx(0);`
	`...`
	`void Phone_OnInitialized(string Msg)`
	`{`
	`        //Verify ‘Msg’ argument`
	`}`

Shutdown	Method destroys SIP stack inside ‘phone’ instance.

Reinitialize	Method initializes back ‘phone’ instance, which was stopped by ‘Shutdown’. ‘Shutdown/ Reinitialize’ were added as workaround for specific cases and also for internal testing. It’s not recommended to use them. In normal implementation app can destroy whole ‘phone’ instance and create it again.

Config	Property returns ‘Config’ interface, which app uses to modify phone settings.
	Syntax:
	`Config phoneCfg = phone.Config();`
	
	Usage example:
	`Config phoneCfg = phone.Config();`
	`phoneCfg.SignallingTransport = SignallingTransportType.eTransportTCP;`
	`phoneCfg.ActivePlaybackDevice  = phoneCfg.get_PlaybackDevice(0);`
	`phoneCfg.ListenPort = 5077;`
	`...`
	`phone.ApplyConfig();`

ApplyConfig	Method reconfigures phone with updated settings. Method works synchronously and may block current thread in case when app added changes, which require to restart SIP stack (like set new `SignallingTransport, ListenPort,` …). Typically SDK applies changes fast enough. Syntax: `phone.ApplyConfig();`
CancelConfig	Method reverts Config changes to its previous state (fixed by last `ApplyConfig`). Syntax: `phone.CancelConfig();`

Make outgoing/receive incoming calls

StartCall	Method starts new outgoing call. It creates new connection, moves it to current (first available) line and sends `SIP INVITE` request using default SIP account. When method fails it raises exception (C#)/returns error code (c++). Syntax: `Phone.StartCall(string destination);` Method arguments: *Destination* - SIP address (extension/phone number) of remote side. Notes: 1. When SDK can't parse destination string it breaks and generates event `OnPhoneNotify`. 2. SDK tries to move created connection to current line. If it's occupied with another call method searches and moves connection to first available line. When all lines are busy (occupied) method breaks. 3. When received SIP message "`SIP 180 Ringing`" SDK raises event `OnRemoteAlerting(n, 180, "Ringing")` and starts plays dial tones (long "beep"). App can disable this feature using property Config.DialToneEnabled=0. SDK stops playing dial tones when received RTP stream from remote side or when call answered. 4. When call answered on remote side SDK raises events `OnEstablishedConnection + OnEstablishedCall`. 5. When call didn't answered during `Config.CallInviteTimeout` seconds SDK cancels this call and raises `OnClearedConnection+OnClearedCall`.
StartCall2	Recommended method for starting outgoing calls. It does same as 'StartCall' and additionally returns `connectionId`, assigned to this call. Syntax: `int connectionId = Phone.StartCall2(destination);`
StartCall3	Method does same as 'StartCall2' and additionally allows override SIP user name in 'From' header, which will be sent in INVITE request. Syntax: `int connectionId = Phone.StartCall3(destination, "FromMe");` It's not recommended to use this method as it may cause call handling issues related to overridden sip user name.

StartCall4	Method does same as 'StartCall2' and additionally allows select which network transport use for sending SIP INVITE request.
	
	Syntax:
	`int connectionId = Phone.StartCall4(destination, TransportType.eTransportTCP);`
	Method requires to configure 'phone' instance with few transports:
	`Phone.Config.SignallingTransport = (int)TransportType.eTransportUDP + (int)TransportType.eTransportTCP;`

StartCallEx	Method does same as ‘StartCall2’ and additionally allows to temporary override existing displayName (caller id), configured for current account, with new value.
	
	Syntax:
	`int connectionId = Phone.StartCallEx(destination, “CallerId”);`
	
	Generated header:
	`From: "CallerId"<sip:user@domain>;tag=xxxxxx`

StartCallExLine	Method does same as ‘StartCallEx’ and additionally allows select line, where to move newly created connection. If specified line is occupied method searches first available line. Syntax: `int connectionId = Phone.StartCallExLine(lineId, address, "CallerId");`
StartCallAcc	Method does same as ‘StartCall2’ and additionally allows select SIP account, from which is required to make this call. Syntax: `int connectionId = Phone.StartCallAcc(address, accountIdx);` As‘accountIdx’ – use value in range `[0...phone.Config. ExSipAccount_Count-1]`
HangUp	Method ends/cancels selected call (specified by connectionId). It sends SIP BYE/CANCEL request. Syntax: `phone.HangUp(int connectionId);` App has wait event `OnClearedConnection`, which means, that remote side has confirmed call ending.
HangUpLastCall	Method ends/cancels all (calls) connections on current line. Syntax: `phone.HangUpLastCall();`
HangUpCallLine	Method ends/cancels all (calls) connections on selected line. Syntax: `phone.HangUpCallLine(lineId);`

AnswerCall	Method answers incoming call, received on current line.
	
	Usage example:
	`AbtoPhone_OnIncomingCall2(string AddrFrom, string AddrTo, int connectionId, int lineId)`
	`{`
	` phone.SetCurrentLine(lineId);//set line, where received new call, as current`
	` phone.AnswerCall();//answer call`
	
	Possible use-case with wrong implementation:
	– App established call on line 1 (which is current);
	– App received new incoming call on line 2;
	– When app invokes ‘`Phone.AnswerCall();`’ it can’t answer call, because current is line 1.

AnswerCallLine	Method answers incoming call on specified line. It doesn’t require to switch between lines, when few calls established.
	
	Usage example:
	`AbtoPhone_OnIncomingCall2(string AddrFrom, string AddrTo, int connectionId, int lineId)`
	`{`
	`Phone.AnswerCallLine(lineId);//answer call on line, where it’s received`
	`Phone.PlayFileLine(“1.wav”, lineId);//play some sound`
	`//continue speaking current call`

AnswerCallConn	Does same as “`AnswerCallLine`”, but as argument receives ‘`connectionId`’.
	
	Usage example:
	`AbtoPhone_OnIncomingCall2(string AddrFrom, string AddrTo, int connectionId, int lineId)`
	`{`
	` Phone.AnswerCallConn(connectionId);`

RejectCall	Method rejects incoming call, received on current line. Usage is same as ‘`AnswerCall`’ – app has invoke `SetCurrentLine` first and after that invoke this method.
RejectCallLine	Method rejects incoming call on specified line. Usage is same as ‘`AnswerCallLine`’.

RejectCallLine2	Method rejects incoming call on specified line and sends user defined status code.
	
	Usage example:
	`AbtoPhone_OnIncomingCall2(string AddrFrom, string AddrTo, int connectionId, int lineId)`
	`{`
	` Phone.RejectCallLine2(lineId, 503);`

Switching between calls, create conference

SetCurrentLine	Method set current line by its id. This operation switches internal mixers gains and user can hear sound of call (connections) on selected line and also these connections can hear user's microphone. Additionally current line work like a context for methods like `AnswerCall/RejectCall/HangUpLastCall` - all these methods will try to work with connections on current line. When this method successfully finished it generates event `OnLineSwiched`. Syntax: `phone.SetCurrentLine(int lineId);` Method arguments: *lineId* – value in range [1..8].
IsLineOccupied	Method returns zero when line is ready to start call. Syntax: `int c = phone.IsLineOccupied(lineId);`
HoldRetrieveCurrentCall	Method holds/retrieves all calls (connections) on current line. <u>Difference between switch current line and hold call</u> When app switches lines with different calls, SDK just modifies mixer gains and user can hear call from current line (remote side can hear user). All calls continue to sends/receive RTP streams, which allows app to play/record sound from these calls. When app puts call on hold it stops send/receive RTP streams, which also doesn't allow to play/record sound of this call. Syntax: `phone.HoldRetrieveCurrentCall();`
HoldRetrieveCall	Method does same as "`HoldRetrieveCurrentCall`" and additionally allows specify lineId. Syntax: `phone.HoldRetrieveCall(int lineId);`
JoinToCurrentCall	Method moves connections of specified line to current line, which creates conference call between them. This feature doesn't require support on server side and hosts conference on user's computer. SDK receives multiple RTP streams, mixes them and sends back. All connections can hear each other and local user. Short instruction how to prepare for invoke this method: A. Answer/Start call on line 1; B. Set current line 2; C. Answer/Start call on line 2; D. Join connection from line 1 to line 2: `phone.JoinToCurrentCall(1);` Syntax: `phone.JoinToCurrentCall(int lineId);`

MoveConnectionToLine	This method does same as “JoinToCurrentCall”, but has different arguments, which allows control this operation easier.
	
	Syntax:
	`phone.MoveConnectionToLine(int lineId, int ConnectionId);`
	
	Example
	`//Move connection ‘112’ to line ‘4’`
	`phone.MoveConnectionToLine(4, 112);`

SetConnectionContribution	Method allows to set mixer gains for selected connection in conference call.
	
	Syntax:
	`phone.SetConnectionContribution(int ConnectionId, int inputGain, int outputGain);`
	
	Example:
	`//Disable input sound of connection 120 – it can only hear`
	`phone.SetConnectionContribution(120, 100, 0);`

SetConnectionContributionRelated	Method sets mixer gains for selected connection in relation to other connection into conference call.
	
	Syntax:
	`phone.SetConnectionContributionRelated(int ConnIdInput, int ConnnIdOutput, int inputGain, int outputGain);`
	
	<u>Usage example:</u>
	App established conference call between: ‘client’, ‘agent’, ‘manager’.
	‘Manager’ can hear both, speak to agent and advise, ‘client’ shouldn’t hear manager.
	
	`phone.SetConnectionContributionRelated(connIdManager, connIdClient, 100, `**0**`).`

SetConnectionContributionRelatedLocal	Method sets mixer gains for selected connection in relation to local side. Syntax: `phone.SetConnectionContributionRelatedLocal(int ConnIdInput, int inputGain, int outputGain);` Notes: For example, conference call, that contains local side and two connections (connectionIdA, connectionIdB). After `SetConnectionContributionRelatedLocal(connectionIdA, 100, 0)` local side can't hear sound of microphone from connectionIdA.

Transfer/redirect calls

TransferCall	Method implements unattended transfer of call established on current line via sending SIP REFER request. When selected line has few connections SDK transfers only first one.
	
	Syntax:
	`phone.TransferCall(string Destination);`
	“`Destination`” – address/extension/phone number of remote side, where to transfer (SDK will copy it to “Refer-To” header.
	
	Usage example:
	1. Established call on line 1, which is current, connectionId=104
	2. App invokes:
	`phone.TransferCall(“555”)`
	3. SDK sends request to remote side
	`SIP REFER`
	…
	`Referred-By: sip:user@domain`
	`Refer-To: sip:555@domain`
	4. Remote server answers with SIP NOTIFY
	5. SDK parses response and raises event:
	`OnPhoneNotify("Redirect Success. Connection: %d Line: %d")`
	`OnPhoneNotify("Redirect Failure. Connection: %d Status: %d Line: %d")`
	6. After success transfer SDK doesn’t end call automatically
	App can implement it in following way:
	`private void AbtoPhone_OnPhoneNotify(string message)`
	`{`
	`    //"Redirect Success. Connection: x";`
	`    //"Redirect Failure. Connection: x Status y";`
	`    Match match = Regex.Match(message, @"Redirect.*Connection: \d+");`
	`    if (match.Success)`
	`    {`
	`        string connIdStr = Regex.Match(match.Value, @"\d+").Value;`
	`        AbtoPhone.HangUp(int.Parse(connIdStr));`
	`    }`

TransferCallLine	Method does same as “TransferCall” and additionally allows to specify line, where is required to make transfer. Syntax: `phone.TransferCallLine(int lineId, string Destination);`
TransferConnection	Method does same as “TransferCall” and additionally allows to specify which call is required to transfer by its `connectionId`. Syntax: `phone.TransferConnection(int ConnectionId, string Destination);`
AttendedTransferCall	Method implements attended (consulting) transfer of call, established on current line. Short instruction how to prepare for invoke this method:

	1. Answer/Start call on line 1
	2. Set current line 2
	3. Answer (establish) second call
	4. Make transfer from line 2 to line 1.
	5. Hang up.
	
	Syntax:
	`phone.AttendedTransferCall(int lineId);`
	
	Usage example:
	1. Established calls: on line 1 (which is current) with extension 401, and on line 2 with extension 402;
	
	2. App invokes:
	`//transfer call from current line to call on line 2`
	`phone.AttendedTransferCall(2);`
	
	3. SDK sends request to remote side
	`SIP REFER`
	`…`
	`Referred-By: sip:user@domain`
	`Refer-To: sip:402@domain?Replaces=Kzpdb7GHxLktR2xXs5lIWA..%3Bto-tag%3DEVIkMVlV.8LINPD.-NNdSwFkxrqpx4WG%3Bfrom-tag%3D8f4d4e17`
	
	4. Remote server answers with SIP NOTIFY
	
	5. SDK parses response and raises event:
	`OnPhoneNotify("Redirect Success. Connection: %d Line: %d")`
	`OnPhoneNotify("Redirect Failure. Connection: %d Status: %d Line: %d")`
	
	6. After success transfer SDK doesn’t end call automatically
	App can implement it in following way:
	`private void AbtoPhone_OnPhoneNotify(string message)`
	`{`
	`    //"Redirect Success. Connection: x";`
	`    //"Redirect Failure. Connection: x Status y";`
	`    Match match = Regex.Match(message, @"Redirect.*Connection: \d+");`
	`    if (match.Success)`
	`    {`
	`        string connIdStr = Regex.Match(match.Value, @"\d+").Value;`
	`        AbtoPhone.HangUp(int.Parse(connIdStr));`
	`    }`

AttendedTransferCallLine	Method does same as “`AttendedTransferCall`” and additionally allows specify between which lineId’s is required to make transfer. Syntax: `phone.AttendedTransferCallLine(int fromLineId, int toLineId);`
RedirectCallLine	Method allows redirect incoming call without answer it. Syntax: `phone.RedirectCallLine(int lineId, string destination);`

Usage example:

1. Add following implementation of incoming call event handler

```
void Phone_OnIncomingCall2(string From, string To, int connId, int lineId)
{
  phone.RedirectCallLine(lineId, "150");
```

2. When incoming call received SDK generates and sends request to remote server:

```
SIP/2.0 302 Moved Temporarily
Contact: <sip:150@domain>
To: <sip:user@domain>;tag=xxxxx
From: <sip:from@domain>;tag=yyyyy
...
```

3. SDK raises `OnClearedConnection + OnClearedCall` events

Play sound from file/buffer

PlayFile	Method starts playing file on current line. Returns non zero when playing started.
	Syntax:
	`int result = phone.PlayFile(string FilePath);`
	"`FilePath`" – path (name) of file, which is required to play.
	<u>Notes:</u>
	1. SDK verifies if playing is already started on specified line and return 0 + raises event [OnPhoneNotify](#)`("File playing has already started")`
	App can set option: `phone.Config.`**`OverridePlayingFileEnabled`**` = true`
	In this mode SDK will stop currently playing file and start new one.
	2. When passed file name only, SDK tries to find this file in folder
	`C:\Users\user\AppData\Local\VoIP Video SIP SDK`
	When file wasn't found in this folder SDK will search it in folder where executable was started.
	When file wasn't found in this folder SDK will return 0 + raise event
	[OnPhoneNotify](#)`("Can't play. File '%s' doesn't exist")`
	When passed empty string it stops playback.
	3. SDK detects extension of specified file. It has to be `*.wav` or `*.mp3`.
	When 'mp3' extension detected SDK will try to decode input file to temporary wav file and play it.
	Decoding implemented via external app "`lame.exe`", which SDK searches in folder with "`SIPVoipSDK.dll`".
	4. SDK has 2 modes of playing files:
	Mixer – in this mode file is playing via mixer input and is available for local side, remote side and recorder, but can't be played few files simultaneously to different connections.
	Connection – in this mode SDK can play few files simultaneously to different connections.
	App can switch modes using property:
	`phone.Config.`**`MixerFilePlayerEnabled`**` = false/true;`
	5. When playing finished/(stopped by app) SDK generates event
	`OnPlayFinished/OnPlayFinished2`.

PlayFileLine	Method does same as "PlayFile" and additionally allows to specify line, where to start playing. Syntax: `phone.PlayFileLine(string FilePath, int lineId)`
BeepFile	Method does same as "PlayFile" and additionally allows specify volume and playing direction. Syntax: `int result = phone.BeepFile(string FilePath, int volume, BeepFileDirection d);` '`volume`' – allows set volume of file player. Value in range [0..100]. Default: 50. '`BeepFileDirection`' – one of: `eLocalOnly, eRemoteOnly, eBoth.`

SeekPlayingFile	Method allows seek position of player on specified line.
	Syntax:
	`phone.SeekPlayingFile(int lineId, int offestMs);`
	Usage example:
	`//Start playing file on line 1`
	`phone.PlayFileLine(“mozart.wav”, 1);`
	…
	`//Move player position to 4500ms from file start`
	`phone.SeekPlayingFile(1, 4500);`

TogglePausePlayingFile	Method allows pause/start file player on specified line. Syntax: `phone.TogglePausePlayingFile(int lineId);`
CalcAudioFileDuration	Method calculates duration (in ms) of specified audio file (wav or mp3). Syntax: `int duration = phone.CalcAudioFileDuration(string FilePath);`
StopPlayback	Method stops playing file on current line. Syntax: `phone.StopPlayback();`
StopPlaybackLine	Method stops playing file on specified line. Syntax: `phone.StopPlaybackLine(int lineId);`
SetPlayingFileContribution	Method set mixer gain of file player. It allows to make player’s sound louder or silent. This value used when app set ‘`MixerFilePlayerEnabled’=true`). Syntax: `phone.SetPlayingFileContribution(int gain);` ‘gain’ – Value in range [0..100]. Default: 50.
SetPlayingFileContributionConn	Method set mixer gain of file player for specified connection only. This value used when app set ‘`MixerFilePlayerEnabled’=false`). Syntax: `phone.SetPlayingFileContributionConn(int gain, int connectionId);` ‘gain’ – Value in range [0..100]. Default: 50.
PlayBuffer	Method starts playing sound from memory buffer to call on current line. SDK makes copy of input buffer and app can free it when method ends. Buffer has contain audio samples as Int16 values. Syntax: `phone.PlayBuffer(long buffer_ptr, int sizeBytes, int rate);` “buffer_ptr” – pointer to input buffer “sizeBytes” – buffer size, bytes “rate” – input audio sample rate. SDK will resample input audio if required. Usage example:

```
//Alloc memory buffer, record sound from mic, play recorded buffer to speaker.
const int recordSeconds = 10;
int TestRecordBufSamples = 0, TestRecordBufBytes=0;
IntPtr TestRecordBuf = IntPtr.Zero;

void recordBufMenuItem_Click(object sender, EventArgs e)
{
  if (TestRecordBuf == IntPtr.Zero)
  {
    TestRecordBufSamples = Phone.Config.SamplesPerSecond * recordSeconds ;
    TestRecordBufBytes = TestRecordBufSamples * Marshal.SizeOf(typeof(Int16));
    TestRecordBuf = Marshal.AllocHGlobal(TestRecordBufBytes);
  }
  Phone.StartRecordingBuffer((long)TestRecordBuf, TestRecordBufBytes);
}

void Phone_OnRecordFinished(string Msg)
{
  if (TestRecordBuf != IntPtr.Zero)
    Phone.PlayBuffer((long)TestRecordBuf, TestRecordBufBytes,
Phone.Config.SamplesPerSecond);
```

Record sound to file/buffer

StartRecording	Method starts recording sound of call(s) on current line.
	Syntax:
	`phone.StartRecording(string FilePath);`
	
	Notes:
	1. Method captures sound from mixer output and recorded file will contain:
	– local (microphone)sound;
	– remote sound (received from call on current line);
	– file player and tones
	When recorded started and app switches lines – SDK will capture sound from call on newly selected line.
	
	2. When app invokes this method and recording has already started - SDK raises exception(C#)/returns error code (c++) with message `"Can't start recording".`
	
	3. When app uses as argument only name of file (without path) SDK stores recorded file in local application data folder:
	`"C:\Users\-USER-\AppData\Local\VoIP Video SIP SDK"`
	
	4. If 'FilePath' argument contains path which doesn't exist SDK can't create it automatically and returns without start recording.
	
	5. By default SDK stores recorded file in uncompressed wav format.
	Option `Config.`**`MP3RecordingEnabled`**`=1/0,` allows enable compressing recorded files to mp3.
	
	6. SDK automatically appends "wav"/"mp3" extension to file name.
	If app requires to use own nonstandard extension, which already included in 'FilePath' argument it can be done via option:
	`Config.`**`AppendExtToRecFileNameEnabled`**` = true/false.`
	
	7. When call ended – SDK stops recording automatically on this line.

MuteRecorder	Method switches mixer gains and causes capturing silence to currently recording file, started by method '[StartRecording'](#). It can be used to prevent saving sensitive data, which remote user says. Syntax: `phone.MuteRecorder(true/false);`
StopRecording	Method stops recoding, which was started by [StartRecording](#). App can access recorded file immediately after invoking this method as SDK may finishing recoding in background thread. To solve this case SDK raises event `OnRecordFinished/ OnRecordFinished2,` which app can handle and access file securely. Syntax: `phone.StopRecording();`

StartRecordingLine	Method does same as “StartRecording” and additionally allows to specify line, where to start recording.
	
	Syntax:
	`phone.StartRecordingLine(int lineId, string FilePath);`
	
	Note:
	Starting from SDK build 2019.09.28 added ability to use few mixer recorders simultaneously.
	Now app can invoke:
	`phone.StartRecordingLine(1, “call1”);`
	`phone.StartRecordingLine(2, “call2”);`
	SDK will start recording two files.

StopRecordingLine	Method does same as “StopRecording”, but app has to use it in pair with “StartRecordingLine”. Syntax: `phone.StopRecordingLine(int lineId);`

StartRecordingConnection	Method starts recording sound from specified connection only.
	
	Syntax:
	`phone.StartRecordingConnection(int ConnectionId, string FilePath);`
	
	Usage example:
	`void AbtoPhone_OnEstablishedConnection(string addrFrom, string addrTo, int connectionId, int lineId)`
	`{`
	`  //record local microphone separately`
	`  phone.StartRecordingConnection(0, "_agent");`
	
	`  //record received sound separately`
	`  phone.StartRecordingConnection(connectionId, "_customer");`
	`…`
	`}`

StopRecordingConnection	Method stops recoding, which was started by `StartRecordingConnection`. Syntax: `phone.StopRecordingConnection(int ConnectionId);`
StartRecordingConnectionBuffer	Method starts recording sound, received from specified connection, to memory buffer provided by app. When buffer is full SDK raises event: `OnRecordFinished/OnRecordFinished2` and stops recording. Syntax: `phone.StartRecordingConnectionBuffer (int connId, long bufPtr, int sizeBytes);`
StartRecordingBuffer	Method starts recording sound of call(s) on current line to memory buffer, provided by app. When buffer is full SDK raises event: `OnRecordFinished/OnRecordFinished2` and stops recording. Syntax: `phone.StartRecordingBuffer(long bufferPtr, int bufferSizeBytes);`

Usage example:

```
const int recordSeconds = 10;
int TestRecordBufSamples = 0, TestRecordBufBytes=0;
IntPtr TestRecordBuf = IntPtr.Zero;

void recordBufMenuItem_Click(object sender, EventArgs e)
{
  if (TestRecordBuf == IntPtr.Zero)
  {
   TestRecordBufSamples = Phone.Config.SamplesPerSecond * recordSeconds ;
   TestRecordBufBytes = TestRecordBufSamples * Marshal.SizeOf(typeof(Int16));
   TestRecordBuf = Marshal.AllocHGlobal(TestRecordBufBytes);
  }

  Phone.StartRecordingBuffer((long)TestRecordBuf, TestRecordBufBytes);
}

void Phone_OnRecordFinished2(int connId, int lineId, long buffer)
{
  if (buffer != TestRecordBuf) return;

  //Copy recorded buf
  byte[] destArr = new byte[TestRecordBufBytes];
  Marshal.Copy(TestRecordBuf, destArr, 0, TestRecordBufBytes);

  //Start recording next chunk
  Phone.StartRecordingBuffer((long)TestRecordBuf, TestRecordBufBytes);

  //Handle recorded chunk
..
}
```

Send DTMF tones

SendTone	Method sends single DTMF tone with duration 200ms to all connections on current line.
	Sending tone by this method includes:
	- Generate tone's sound, play it locally and send sound to remote side;
	App can disable playing tones locally via option “`phone.Config.LocalTonesEnabled=0`”
	- Send RTP signaling packets by RFC4733 (RFC2833);
	This way of sending tones will not work if app removed codec “RFC4733 DTMF tones” from codecs list.
	
	Syntax:
	`phone.SendTone(string Tone);`
	“Tone” – symbol of tone, which is required to send.
	Allowed symbols are: `0,1,2,3,4,5,6,7,8,9,*,#, A,B,C,D;`
	
	Example:
	`phone.SendTone(“1”);`
	
	Notes:
	1. App can send next tone only when previous tone already sent.
	2. When SDK received DTMF tone from remote side it raises event: `OnToneReceived`.
	3. App can use this method for generation special tone like: busy, fastbusy, callwaiting.
	Usage example: outgoing call ended with status code ‘486’ – app plays busy tone.
	`void Phone_OnClearedCall(string Msg, int status, int lineId)`
	`{`
	`  if (status == 486)`
	`  {`
	`    AbtoPhone.SendTone("SPECIAL_busy");//play busy tone locally`
	`  }`
	`}`

SendToneEx	This method sends single DTMF to all connections into current line and allow specify sending options more detailed.
	
	Syntax:
	`phone.SendToneEx(`
	` int Tone, int Duration,`
	` int bSendAudio_InBand, int bSend_RFC4733_OutOfBand, int bSend_SIP_INFO);`
	
	“`Tone`” – tone to send.
	Allowed values are: ‘`0’,’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,`
	`’*’,’#’,‘A’,’B’,’C’,’D’`
	“`Duration`” – duration of generated tone in ms.
	“`bSendAudio_InBand`” – if this value is not 0 SDK will generate sound of tone and play it on local and remote side;
	“`bSend_RFC4733_OutOfBand`” – if this value is not 0 SDK will send tone as RTP signaling packets;
	“`bSend_SIP_INFO`” – if this value is not zero SDK will send tone as SIP INFO request.
	
	See also more details in “`SendTone`” method explanation.

SendToneExLine	This method does same as “SendToneEx” and additionally allows specify line/connection where to send tone.
	Syntax:
	`phone.SendToneExLine(`
	` int lineId, int connectionId,`
	` int Tone, int Duration,`
	` int bSendAudio_InBand, int bSend_RFC4733_OutOfBand, int bSend_SIP_INFO);`
	“`connectionId`” – when this argument is not 0 – SDK will send tone to specified connection only (value of “lineId” is ignored in this case).
	“`lineId`” – when this argument is not 0 – SDK will send tone to all connections of specified line. In case of wrong value – tone will be sent to connections on current line.

SendToneExLine2	This method does same as “SendToneExLine” and additionally allows to send sequence of tones.
	Syntax:
	`phone.SendToneExLine2(`
	` int lineId, int connectionId,`
	` string Tone, int Duration,`
	` int bSendAudio_InBand, int bSend_RFC4733_OutOfBand, int bSend_SIP_INFO);`
	Usage example:
	`phone.SendToneExLine2(0, 104, “124#”, 200, 0, 1, 0);`
	This line of code send sequence of tones “124#” to connection, which id is ‘104‘, using RTP signaling packets. Duration of each tone in sequence is 200ms.

StopToneLine	Method stops sending DTMF tone on specified line. Syntax: `phone.StopToneLine(int lineId)`
SetTonesContribution	This method allows set mixer gain (volume) of generated tones sound. Syntax: `phone.SetTonesContribution (int gain)` “gain” – value in range [0..100]. Default: 50.

Send Baudot (RTTY) codes

	Here are links with more information about feature: https://www.sigidwiki.com/wiki/Radio_Teletype_(RTTY) https://en.wikipedia.org/wiki/Baudot_code

SendBaudotTone	Method generates sound of specified Baudot tone, plays it locally and sends to remote side of call, established on current line.
	
	Syntax:
	`phone.SendBaudotTone(int BaudotTone);`
	
	Usage example:
	`phone.SendBaudotTone('1');`
	
	Notes:
	When SDK finished sending baudot tone (string) it raises event: `OnBaudotFinished.`
	
	SDK has setting `Config.TonesBaudRate,` which allows to specify baud sped.
	Value should be one of: `50.0, 47.0, 45.0, 45.5.`
	
	When is required to recognize baudot tones from received sound app has invoke
	`Phone.Config.TonesTypesToDetect = (int)ToneType.eToneBaudot;`
	`Phone.ApplyConfig();`
	SDK sends recognized tones to app via raising events: `OnBaudotToneReceived.`

SendBaudotTone2	Method does same as "`SendBaudotTone`" and additionally allows specify volume of generated sound. Syntax: `phone.SendBaudotTone2(int BaudotTone, int gain);` gain – value in range [0..100], default – 50.

SendBaudotString	Method does same as "`SendBaudotTone`" and allows specify sequence of tones instead of single tone.
	
	Syntax:
	`phone.SendBaudotString(string strToSend);`
	
	Usage example:
	`phone.SendBaudotString("The quick brown fox jumps over the lazy dog");`

SendBaudotString2	Method does same as "`SendBaudotString`" and additionally allows specify volume of generated sound. Syntax: `phone.SendBaudotString2(string strToSend, int gain);`

Send SIP requests (MESSAGE/INFO/NOTIFY/REFER)

SendTextMessage	Method sends text message as `SIP MESSAGE` request. Syntax: `phone.SendTextMessage(string Address, string Message, int bSendUnicode);` “Address” – destination (SIP URI/extension); “Message” – text message, which is required to send. SDK receives this argument as Unicode string and converts to utf-8. “bSendUnicode” – this argument is not using by SDK. Notes: 1. App can detect was message sent successfully using event `OnTextMessageSentStatus`, which SDK raises when received “200 OK” response or after timeout. 2. To receive MESSAGE from remote side use event: `OnTextMessageReceived` 3. Some SIP servers doesn’t support sending SIP MESSAGE request between clients. Alternative option, which allows app send some data to same app on remote side – use SIP NOTIFY request, which SDK can send using method `SendRequestNotify`.

SendTextMessage2	Method does same as “`SendTextMessage`” and additionally allows set value of header. ‘Content-Type’.
	
	Syntax:
	`phone.SendTextMessage2(string Address, string Message, string contentType);`
	
	Example:
	`phone.SendTextMessage2(“777”, “Hello”, “text/plain”);`

SendRequestInfo	Method sends in-dialog (app can send it only when call established) SIP INFO request with custom ‘contentType’ header and body.
	
	Syntax:
	`phone.SendRequestInfo(int connectionId, string contentType, string msg);`
	
	Usage example:
	`phone.SendRequestInfo(connId, "application/test", "test body text");`

SendRequestNotify	Method sends does the same as “SendRequestMWINotify”, but uses SIP header “Content-Type: text/plain”. Syntax: `phone.SendRequestNotify(string Destination, string body);`

SendRequestMWINotify	Method sends SIP NOTIFY request to specified destination address with specified body.
	
	Syntax:
	`phone.SendRequestMWINotify(string destination, string body);`
	
	Example:
	This line of code
	`phone.SendRequestMWINotify(“150”, “test”);`
	will send following request:

	NOTIFY sip:150@domain SIP/2.0 Via: SIP/2.0/UDP 172.30.30.129:5080;branch=z9hG4bK-524287-1--- 2c37570a8573570f;rport Contact: <sip:217@172.30.30.129:5080> To: <sip:150@domain> From: <sip:217@domain>;tag=19583c27 Call-ID: cjPYNOvAJT_BYcM7rJ_7dw.. CSeq: 1 NOTIFY Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, SUBSCRIBE, UPDATE, INFO, MESSAGE Content-Type: application/simple-message-summary Supported: replaces, norefersub, answermode, tdialog User-Agent: ABTO SIP SDK Subscription-State: active Event: message-summary Content-Length: 4 test

SendRequestRefer	Method sends SIP REFER request to specified destination address with specified body.
	
	Syntax:
	`phone.SendRequestRefer(string Destination, string body);`
	
	Example:
	This line of code
	`phone.SendRequestRefer(“150”, “test”);`
	will send following request:
	REFER sip:150@domain SIP/2.0
	Via: SIP/2.0/UDP 172.30.30.129:5080;branch=z9hG4bK-524287-1---765ff65d0d21fc32
	Contact: <sip:217@172.30.30.129:5080>
	To: <sip:150@domain>
	From: <sip:217@domain>;tag=7b114d40
	Call-ID: KeWahnjqkYkUBTjp185bgg..
	CSeq: 1 REFER
	Accept: application/x-cisco-remotecc-response+xml
	Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, SUBSCRIBE, UPDATE,
	INFO, MESSAGE
	Content-Disposition: session;handling=required
	Content-Type: application/x-cisco-remotecc-request+xml
	User-Agent: ABTO SIP SDK
	Referred-By: <sip:217@172.30.30.129:5080>
	Content-Length: 4
	
	test

SendRingingMessage	Method sends in-dialog (to remote side of established call, specified by its connectionId) “180 Ringing” request. App can disable automatic sending “180 Ringing” using property “`Config.SendRingingMsgEnabled=0`” and send these requests manually. Syntax: `phone.SendRingingMessage(int ConnectionId);`

Add own SIP headers/Get SIP header value from received request

GetSIPHeaderValue	Method parses last received SIP request on current line and returns value of requested header or empty string.
	
	Syntax:
	`string val = phone.GetSIPHeaderValue(string Header);`
	
	Usage example:
	`private void OnIncomingCall(string addrFrom, int lineId)`
	`{`
	`  string callId = phone.GetSIPHeaderValue(“Call-ID”);`
	` …`

GetSIPHeaderValueLine	Method does same as '`GetSIPHeaderValue`' and additionally allows specify lineId, for Syntax: `string val = phone.GetSIPHeaderValueLine(string Header, int lineId);`
GetSIPMessageLine	Method returns last received SIP request on specified line as string. App can parse it manually and extract required data. Syntax: `string sipMsg = phone.GetSIPMessageLine (int lineId);`

HasSIPBodyVideoMedia	Method parses last receive INVITE request and returns not 0 when SDP body of request contains ‘video’ media. It allows to detect is received incoming call audio only or audio+video.
	
	Syntax:
	`int isVideoCall = phone.HasSIPBodyVideoMedia(int lineId);`
	
	Usage example:
	`private void OnIncomingCall(string addrFrom, int lineId)`
	`{`
	`  int isVideoCall = phone.HasSIPBodyVideoMedia(lineId);`
	` …`

SetSIPHeaderValue	Method adds custom headers to outgoing SIP requests. Configured header values temporary stored in internal variable and removed when SDK sends SIP request.
	
	Syntax:
	`phone.SetSIPHeaderValue(string Header, string value);`
	
	Usage example:
	`private void customStartCall(string dest)`
	`{`
	`  phone.SetSIPHeaderValue(“X-CUSTOM”, “XValue”);`
	`  phone.StartCall(dest);`
	`}`

Select (switch) audio devices / Set volume

	App can select local speaker/microphone devices using properties: `Config.ActivePlaybackDevice, Config.ActiveRecordDevice.` App can use these properties in any time, even when call is established. SDK will work well on computer without audio devices at all. In case when app requires only play/record audio to established calls we suggest to disable local audio subsystem using property: `Config.LocalAudioEnabled=0`.

PlaybackVolume	Method gets/sets sound level of speaker device, selected via setting `Config.ActivePlaybackDevice`.
	
	Syntax:
	`phone.PlaybackVolume = v;`
	Input argument should be in range [0..100]
	
	<u>Notes:</u>
	SDK has 2 different volume implementations, which app can switch using setting:
	`Config.IntenalVolumeImplEnabled.`
	a. “`IntenalVolumeImplEnabled=true`” – this is default mode.
	SDK uses own implementation and modifies value of audio samples before send them to Speaker device.
	
	In this mode volume level means:
	0 – mute speaker.
	1 – set volume to 25% from its original level
	50 – set volume to 100% from its original level
	100 – set volume to 400% from its original level
	
	b. “`IntenalVolumeImplEnabled=false`”. In this mode SDK modifies volume using Window Mixer API.

PlaybackVolumeLine	Method gets/sets sound level of speaker device, selected via setting “`Config.SetPlaybackDeviceLine`”. Syntax: `phone.set_PlaybackVolumeLine(int lineId, int volume);` See more details about implementation – “PlaybackVolume”.
PlaybackMuted	Method mutes/unmutes speaker selected via setting `Config.ActivePlaybackDevice`. Syntax: `phone.PlaybackMuted = 1;//0;`
RecordVolume	Method gets/sets sound level of microphone device, selected via setting `Config.ActiveRecordDevice`. Syntax: `phone.RecordVolume = v;` See more details about implementation – “PlaybackVolume”.

RecordVolumeLine	Method gets/sets sound level of microphone device, selected via setting "Config.SetRecordDeviceLine". Syntax: phone.set_PlaybackVolumeLine(int lineId, int volume); See more details about implementation – "PlaybackVolume".
RecordMuted	Method mutes/unmutes microphone device, selected via setting Config.ActiveRecordDevice. Syntax: phone.RecordMuted = 1;//0;
RingerVolume	Method gets/sets sound level of speaker device, selected via setting "Config.RingerDevice". Syntax: Phone.RingerVolume = 75; See more details about implementation – "PlaybackVolume".
RingerMuted	Method mutes/unmutes speaker device, selected via setting Config.RingerDevice. Syntax: phone.RingerMuted = 1;//0;
BindToAudioDevices	Method obsolete.
RestartAudioSubsystem	Method is usable for fast handling changes with audio devices. For example – when audio device unplugged/plugged back app can just invoke this method and it will internally update list of available audio devices and selects first available one (when 'ActivePlaybackDevice'/' ActiveRecordDevice' missed)/selects previously configured device. App can invoke this method in any time, even during a call. Syntax: phone.RestartAudioSubsystem()
RestartAudioSubsystemEx	Method does same as 'RestartAudioSubsystem' and additionally allows to set primary and secondary audio device. When primary device missed – SDK will try to select secondary one. Syntax: phone.RestartAudioSubsystemEx(string PrimaryPlaybackDvc, string SecPlaybackDvc, string PrimaryRecordDvc, string SecRecordDvc);
SyncDevicesList	Method allows to detect newly connected/disconnected audio devices and manually select required one. It just updates list of audio devices, available through properties: Config.PlaybackDeviceCount/get_PlaybackDevice, Config.RecordDeviceCount/ get_RecordDevice. Syntax: phone.SyncDevicesList();

Select (switch) network interfaces

	SDK allows select local network interface using property: `Config.ActiveNetworkInterface.` via assigning its value as one of . Value 'Auto' means that SDK will try to automatically select which local network interface to use (bind SIP/RTP sockets and send IP address of this interface in SIP headers). Implementation uses method GetBestRoute and detect which network interface can send packets to remote 'RegDomain'/'RegProxy' address (domain). In case when app doesn't set these properties SDK uses first found network interface.
RestartNetworkSubsystem	Method restarts SIP stack and binds to newly available network interface. It's usable in cases when computer switched between WiFi networks or lost and restored network connection. Syntax: `phone.RestartNetworkSubsystem(bool bForce);` "bForce" – when this value is 'true' SDK restarts whole SIP stack and audio subsystem. When this value is 'false" – SDK will verify is current local address, resolved automatically, same as previous one and restart SIP stack when address is different.

Video call functionality

	SDK can make and receive video calls even when local video device missed. To activate this feature use option: `Config.VideoCallEnabled=1`. To select local video device use option: `Config.ActiveVideoDevice`. To control local video, which SDK captures and sends to remote side use options: `Config.VideoFrameWidth/VideoFrameHeight/VideoFrameRate/VideoBitRate`.

AssignVideoWindow	Method configures SDK to render video stream, received from connection, specified by its 'connectionId', on window specified by its hwnd.
	By default SDK renders received video on window, specified via property `Config.RemoteVideoWindow`. This method is usable in case when app established few video calls and is required to assign window for each of them.
	
	Syntax:
	`phone.AssignVideoWindow(int ConnectionId, int VideoWindowHwnd);`
	
	Usage example:
	`void AbtoPhone_OnEstablishedConnection(string addrFrom, string addrTo, int`
	`connectionId, int lineId)`
	`{`
	`  phone.AssignVideoWindow(connectionId, pictureReceivedVideo.Handle);`

MuteLocalVideo	Method stops/starts sending local video RTP stream. Syntax: `phone.MuteLocalVideo(int bMute);` "bMute" – when value is not 0 SDP stops sending own video, when value is 0 – SDK starts sending video. Note: SDK has option `Config.VideoAutoSendEnabled` which allows to disable sending own video when video call started/answered. App can start sending video later, using this method: `phone.MuteLocalVideo(0);`

StartVideoRecording	Method starts recording video, received from connection, specified by its 'ConnectionId' to local file.
	
	Syntax:
	`phone.StartVideoRecording(int ConnectionId, string FilePath);`
	"ConnectionId" – id of connection, which video stream is required to record.
	0 – means local video device.
	"FilePath" – file name or path\filename, where to store recorded video file.
	
	Usage example:
	`void Phone_OnEstablishedConnection(string from, string to, int connectionId,`
	`int lineId)`
	`{`
	`   phone.StartVideoRecording(connectionId, connectionId.ToString());`
	
	Notes:
	See more notes about recording: StartRecording.

StopVideoRecording	Method stop recording of video file, started by: ‘StartVideoRecording’. Syntax: phone.StopVideoRecording(int ConnectionId);
StartPlayVideoFile	Method allows to override local video input. It stops current video source, reads video frames from specified file and send them to remote side. Before send frames from file SDK applies same resolution/frame rate and bitrate setting. Syntax: phone.StartPlayVideoFile(string FilePath);
StopPlayVideoFile	Method stops playing video from file, which was started by ‘StartPlayVideoFile’ and restores capturing from camera (if it was configured). Syntax: phone.StopPlayVideoFile();
StartScreenSharing	Method allows to override local video input. It stops current video source, starts capturing desktop image and send to remote side of established call. Before send frames SDK applies same resolution/frame rate and bitrate setting. Syntax: phone.StartScreenSharing(int bStart); “bStart” – argument is ignored by SDK.
StopScreenSharing	Method stops capturing image from desktop, which was started by ‘StartScreenSharing’ and restores capturing from camera (if it was configured). Syntax: phone.StopScreenSharing();
GetReceivedVideoFrames	Method returns number of video frames, received and decoded from remote side, specified by ‘ConnectionId’. It allows to detect is remote side sending some video or not. Syntax: int frames = phone,GetReceivedVideoFrames(int ConnectionId);
GetVideoFrame	Method copies last received and decoded video frame to provided memory buffer. Frame format is RGB32. To avoid possible memory corruption, caused by small buffer app has to use method ‘GetVideoFrameSize’, receive resolution and alloc buffer with size: 4*width * height. Method allows implement custom video render, which doesn’t require to create window and get its hwnd. Syntax: phone.GetVideoFrame(int ConnectionId, long bufferPtr, int bufSizeBytes); “bufferPtr” – pointer to memory buffer. “bufSizeBytes” – buffer size, bytes.
GetVideoFrameSize	Method returns resolution of received video frame. Syntax: phone.GetVideoFrameSize(int ConnectionId, ref int width, ref int height);

Get 'phone' state/version/path

RetrieveExternalAddress	Method returns resolved IP address, which SDK uses when generates SIP requests/responses. Syntax: `String addr = phone.RetrieveExternalAddress();`
RetrieveVersion	Method returns version (build date) of currently loaded SDK. It helps to detect which exactly SDK version is using after updating SDK or install/uninstall different setup. Syntax: `String addr = phone.RetrieveVersion();`
SDKPath	Method returns path, from which was loaded SDK dll. It helps to detect folder, where is registered SDK dll and also verify was it loaded from proper location. Syntax: `String addr = phone.SDKPath();`

MakeDumpFile	Method creates mini-dump file and stores it as:
	`<phone.Config.LogPath>\crash_HHMMSS_<SDK_VERISON>.dmp`
	
	It's usable in case when detected some wrong case (like freezing threads), but app/SDK continues running and allows developers to detect reason of freezing or other issues.
	
	Syntax:
	`phone.MakeDumpFile();`

IsPhoneBusy	Method returns 0 if not found any pending connections (dialing/answering/disconnecting). Syntax: `int r = AbtoPhone.IsPhoneBusy();`

GetConnectionStat	Method returns RTCP statistic of call, specified by it connectionId as string in json format.
	
	Syntax:
	`string stat = phone.GetConnectionStat(int connectionId);`
	
	Usage example:
	`string stat = phone.GetConnectionStat(connId);`
	
	Example of string, returned by this method (it has json format):
	`{`
	`" RX":{"packets":1729, "bytes":276640, "loss":0, "p_loss":0.000,`
	`"jt_min":0.226, "jt_mean":3.573, "jt_max":3.605,`
	`"ssrc":"59A7E820","addr":"172.30.30.150:17384"},`
	`" TX":{"packets":1695, "bytes":271200, "loss":0, "p_loss":0.000,`
	`"jt_min":0.000, "jt_mean":0.000, "jt_max":3.605,`
	`"ssrc":"59AF1FED","addr":"172.30.30.129:17384"},`
	`" RTT": {"min":0.000, "mean":0.000, "max":0.000 }`
	`}`
	`RX - received RTP stream`
	`TX - sent RTP stream`

AbtoPhone Subscriptions

Subscriptions	This property returns Subscriptions interface, with methods, which allows to send SIP SUSCRIBE/SIP NOTIFY requests. Syntax: `Subscriptions s = phone.Subscriptions;`

SubscribeBLF	Method sends SIP SUBSCRIBE request with headers:
	`“Event: dialog”`
	`“Accept: application/dialog-info+xml”`
	and returns unique ‘subscriptionId’.
	
	Syntax:
	`int SubscribeBLF(string sipUri);`
	
	Usage example:
	`int subscriptionId = phone.Subscriptions.SubscribeBLF("150");`
	This code generates and sends request:
	`SUBSCRIBE sip:150@domain SIP/2.0`
	`Via: SIP/2.0/UDP 172.30.30.129:5080;branch=z9hG4bK-524287...`
	`Max-Forwards: 70`
	`Contact: <sip:217@172.30.30.129:5080>`
	`To: <sip:150@domain>`
	`From: <sip:user@domain>;tag=625e1832`
	`Call-ID: 0xYVHnd-__O3cYJQGx205g..`
	`CSeq: 1 SUBSCRIBE`
	`Expires: 3600`
	`Accept: application/dialog-info+xml`
	`Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, ...`
	`Supported: replaces, norefersub, answermode, tdialog`
	`User-Agent: ABTO SIP SDK`
	`Event: dialog`
	`Content-Length: 0`
	
	When remote side, specified by its ‘sipUri’ is not available or rejects this subscription SDK raises event: `OnSubscribeStatus(subscriptionId, int statusCode, string statusMsg)`
	For example: `OnSubscribeStatus(1, 480, “Terminated”).`
	
	When remote side accepts subscription it answer 200OK + NOTIFY.
	SDK handles NOTIFY and raises event: `OnSubscriptionNotify.`
	Same even is raised when BLF/Presence state of remote app was changed and it notifies app about this.
	
	Each 3600 seconds SDK automatically updates subscriptions (tries to re-send same request, when previous wasn’t answered).
	App can modify update timeout using property: `Config.SubscriptionExpire.`
	Example: `phone.Config.SubscriptionExpire = 2400;`

UnSubscribeBLF	Method removes existing subscription via sending SIP SUBSCRIBE request with header ‘`Expires: 0`’. Syntax: `void UnSubscribeBLF(int subscriptionId);` “subscriptionId” – id, returned by method ‘SubscribeBLF’. When received answer from remote side SDK raises event ‘`OnSubscribeStatus`’.

SubscribePresence	Method does same as “SubscribeBLF”, and send SIP headers:
	`“Event: presence”`
	`“Accept: application/pidf+xml”.`
	
	Syntax:
	`int SubscribePresence(string sipUri);`
	
	Usage example:
	`int subscriptionId = phone.Subscriptions.SubscribePresence("150");`

UnSubscribePresence	Method does same as “UnSubscribeBLF”. Syntax: `Void UnSubscribePresence(int subscriptionId);`

SubscribeMessageSummary	Method does same as “SubscribeBLF”, and sends SIP headers:
	`“Event: message-summary”`
	`“Accept: application/simple-message-summary”`
	and as destination uses own SIP user name.
	It allows to subscribe receiving voice mail notifications from server side.
	
	Syntax:
	`int SubscribeMessageSummary();`

UnSubscribeMessageSummary	Method does same as “UnSubscribeBLF”. Syntax: `void UnSubscribeMessageSummary (int subscriptionId);`

SubscribeCustomEvent	Method does same as “SubscribeBLF”, and sends in SIP header with user defined values:
	`“Event: [event]”`
	`“Accept: application/[mimeSubType]”.`
	
	Syntax:
	`int SubscribeCustomEvent(string event, string mimeSubType, string sipUri);`

UnSubscribeCustomEvent	Method does same as “UnSubscribeBLF”. Syntax: `void UnSubscribeCustomEvent(int subscriptionId);`
Notify	Method does nothing.
NotifyBLF	Method does nothing.
Accept	Method does nothing.
Reject	Method does nothing.
NotifyPresence	Method sends SIP NOTIFY request to all clients, subscribed on current present state of currently registered user. Syntax: `void NotifyPresence(int open, string status);` “open” – when this argument is not 0 SDK adds status `<basic>`**`open`**`</basic>`” to body of generated request (see below).

“status” – status string, which is required to send.

Usage example:

```
phone.NotifyPresence(1, “Ready for chat”);
```

This code generates and send following request:

```
NOTIFY sip:217@domain SIP/2.0
Via: SIP/2.0/UDP 172.30.30.129:5081;branch=z9hG4bK-524287-1---
fd0fb97e010b5733;rport
Max-Forwards: 70
Contact: <sip:218@172.30.30.129:5081>
To: <sip:217@172.30.30.150>;tag=5f52bc47
From: <sip:user@domain>;tag=bc47d51c
Call-ID: rrlzvYZnGXJ-n81I4ZbzAw..
CSeq: 5 NOTIFY
Content-Type: application/pidf+xml
User-Agent: ABTO SIP SDK
Subscription-State: active;expires=3566
Event: presence
Content-Length: 253

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          entity="sip:217@172.30.30.150">
  <tuple id="8b4cab66" >
     <status><basic>open</basic></status>
     <note>Ready for chat</note>
  </tuple>
</presence>
```

AbtoPhone Events

Name	Description

OnInitialized	SDK raises this event when initialization completed.
	Syntax:
	`void OnInitialize(string Msg);`
	Possible values of “Msg” string are:
	`“Initialized” – SDK initialized successfully;`
	`“No output audio devices present!”`
	`“No input audio devices present!”`
	`“No network interfaces present!”`
	`“No codecs found!”`
	`“License key not set” etc.`

OnPhoneNotify	SDK raises this event when has some additional information about event or phone state.
	Syntax:
	`void OnPhoneNotify(string Msg);`
	Example:
	After generation OnRegistered event, SDK also generates this event with message:
	`“Registered as user@domain”`

OnLineSwiched	SDK raises this event when app switched current line using method `phone.SetCurrentLine.` Syntax: `void OnLineSwiched(int lineId);` *lineId* – value in range [1..8].
OnEstablishedConnection	SDK raises this event when received incoming or answered outgoing connection. Syntax: `void OnEstablishedConnection(string addrFrom, string addrTo, int connectionId, int lineId);` *addrFrom* – value, received in SIP header 'From'. *addrTo* – value, received in SIP header 'To'. *connectionId* – unique Id, assigned to this connection. *lineId* – index of line (in range 1..8), which contains this connection.
OnClearedConnection	SDK raises this event when connection ended. Syntax: `void OnClearedConnection(int connectionId, int lineId);` *connectionId* - connection unique Id. *lineId* - index of line (in range 1..8), which contains this connection.
OnClearedConnection2	Event is raised in same time with “OnClearedConnection” and has one more argument. Syntax: `OnClearedConnection2(int connectionId, int lineId, int status)` *Status* – status code, received from remote side.

OnIncomingCall	SDK raises this event when incoming call received. Syntax: `OnIncomingCall(string addrFrom, int lineId);` *AddrFrom* – value, received in SIP header 'From'. *lineId* – index of line (in range 1..8), which contains this connection.
OnIncomingCall2	Event is raised in same time with "`OnIncomingCall`" and is more usable as has one more argument – 'ConnectionId '. Syntax: `OnIncomingCall2(string addrFrom, string addrTo, int connectionId, int lineId);`
OnEstablishedCall	SDK raises this event when call successfully established (sent `200 OK` and received `ACK`). Syntax: `OnEstablishedCall(string BSTR Msg, int LineId);` *Msg* - typical is equal "`Call Established on Line #1`". *lineID* - index of line (in range 1..8) where received this connection.
OnClearedCall	SDK generates this event when ended last connection on line (after event `OnClearedConnection`). Syntax: `OnClearedCall(string msg, int status, int lineId);` Event arguments: msg - typical is equal "`Call completed`". status - status code. lineId - index of line (in range 1..8) where cleared this connection.
OnVolumeUpdated	App can use this even to render sound level of speaker/microphone on progress bar or other control. When app set option: `phone.Config.VolumeUpdateSubscribed = 1;` SDK will raise this event each time when speaker/microphone signal level was changed. Syntax: `OnVolumeUpdated(int IsMicrophone, int Level);` Event arguments: *IsMicrophone* - is equal to 1, when detect new signal level from microphone, and 0 - in other case. *Level* - new level value in range [0…32767].
OnRegistered	This event allows control registration state (successful, failed or removed). It's raised when received answer on registration request or didn't receive any answer during some period of time. Syntax: `OnRegister(string msg);`

	Event arguments:
	msg - argument string has following format: "`MSG: CODE REASON ExAccountIdx:x`".
	Where:
	`MSG` – Status string, generated by SDK;
	`CODE` - Status-Code received with SIP response;
	`REASON` - Status-Line received from SIP response;
	`ExAccountIdx` – This part of string is added only when app added few SIP accounts and means index of account, which received this message.
	
	Example:
	`"Registration success: 200 OK ExAccountIdx:1";`
	`"Registration failure: 401 Unauthorized".`
	`"Registration removed: 200 OK".`

OnUnRegistered	SDK never raises this event.

OnPlayFinished	SDK raises this event when finished playing sound file or memory buffer.
	
	Syntax:
	`OnPlayFinished(string msg);`
	
	Event arguments:
	msg - argument string has following format:
	`"Play Finished on Line: <lineIdx>".`
	Example: "Play Finished on Line: 1".

OnPlayFinished2	Event is raised in same time with '`OnPlayFinished`' and is more usable as has argument 'lineId'. Syntax: `OnPlayFinished2(string msg, int lineId)` lineId – line, where finished playing;

OnBaudotFinished	SDK raises this event when finished playing baudot tone started by one of methods:
	`SendBaudotTone, SendBaudotString`
	
	Syntax:
	`OnBaudotFinished(string msg, int lineId);`

OnToneReceived	SDK raises this event when received DTMF tone from remote side. Syntax: `OnToneReceived(int tone, int connectionId, int lineId);` Event arguments: `tone` – received tone; `connectionId` – connectionId from which received this tone; `lineId` – line, which contains this connection; Possible tones are: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '#', 'A', 'B', 'C', 'D', 'E', 'F'.

OnToneDetected	SDK is able to detect some specific signals/tones in received sound.
	By default detector is disabled and app can activate it using property:
	`Phone.Config.TonesTypesToDetect = x;`
	As value of this setting set sum of:
	• ToneType::**eToneDtmf**=1 - enable DTMF tones detection (de default SDK receievs DTMF as RTP signalling - RFC4733)
	• ToneType::**eToneBaudot**=2 - enable Baudot tones detection
	• ToneType::**eToneSIT**=4 - enable SIT tones detection (https://en.wikipedia.org/wiki/Special_information_tone)
	• ToneType::**eToneMF**=8 - enable detection tones with single frequency.
	When detected one of these signals SDK raises event:
	`OnToneDetected(ToneType type, string tone, int connectionId, int lineId)`
	Event arguments:
	type – type of detected signal (one of members 'ToneType' enum);
	tone – detected signal;
	connectionId – id of connection, sent this signal;
	lineId – line, which contains this connection;
	Possible values of 'tone' argument:
	
	When type is '**eToneDtmf**':
	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #, A, B, C, D, E;
	
	When type is '**eToneBaudot**:
	Alphabet characters, digits…
	
	When type is '**eToneSIT**':
	"IC_SIT ";
	"RO'_SIT "
	"RO'_'_SIT”
	"IO_SIT "
	"VC_SIT "
	"NC'_SIT "
	"NC'_'_SIT
	
	When type is '**eToneMF**:
	"450 ";
	"480 ";
	"620 ";
	"AnsweringMachine T-Mobile 1250 ";
	"AnsweringMachine Verizon 1500 ";
	"AnsweringMachine Sprint/Nextel 1400 ";
	"AnsweringMachine AT&T/Cingular 1700 ";
	"FAX 2100";
	"FAX 1100";
	"Reorder ";
	"NorthAmericanBusySignal";
	"UndefFrequency";

OnBaudotToneReceived	SDK raises this event in same time with “OnToneDetected” with same arguments: `OnBaudotToneReceived(int tone, int connectionId, int lineId)` Event arguments: `tone` – detected signal; `connectionId` – id of connection, sent this signal; `lineId` – line, which contains this connection;

OnTextMessageReceived	SDK raises this event when received SIP MESSAGE request. OnTextMessageReceived(string address, string body); Event arguments: address - value received in SIP header 'From'; body – body of received request; SDK is able to properly decode utf-8 strings;
OnReceivedSipNotifyMsg	SDK raises this event when received out-of-dialog SIP NOTIFY request. OnReceivedSipNotifyMsg(string notifyMsg) Event arguments: notifyMsg – received SIP request encoded to string;
OnReceivedSipNotifyMsgInDialog	SDK raises this event when received in-dialog SIP NOTIFY request. OnReceivedSipNotifyMsgInDialog(int connectionId, int lineId, string notifyMsg) Event arguments: connectionId – id of connection, sent this signal; lineId – line, which contains this connection; notifyMsg – received SIP request encoded to string;
OnReceivedRequestInfo	SDK raises this event when received in-dialog SIP INFO request. OnReceivedRequestInfo(int connectionId, int lineId, string contentType, string body); Event arguments: connectionId – id of connection, sent this signal; lineId – line, which contains this connection; contentType – value, received in header ‘ContentType’; body – body of received request;
OnReceivedRequestUpdate	SDK raises this event when received in-dialog SIP UPDATE or SIP INVITE request. OnReceivedRequestUpdate(int connectionId, int lineId, string sipMsg) Event arguments: connectionId – id of connection, sent this signal; lineId – line, which contains this connection; sipMsg – received SIP request encoded to string;
OnReceivedRequestRefer	SDK raises this event when received in-dialog SIP REFER request. OnReceivedRequestRefer(int connectionId, int lineId, string sipMsg) Event arguments: connectionId – id of connection, sent this signal; lineId – line, which contains this connection; sipMsg – received SIP request encoded to string;

OnRemoteAlerting	SDK raises this event when received one of early SIP requests:
	`100 Trying`
	`180 Ringing`
	`183 Session progress`
	Syntax:
	`OnRemoteAlerting(int connectionId, int responseCode, string reason);`
	
	Event arguments:
	`connectionId` - id of connection, sent this request;
	`responseCode` - value from SIP header 'StatusCode'.
	`reason` - value from SIP header 'StatusLine'.
	
	Example:
	When SDK makes outgoing call and received “100 Trying” it will raise
	`OnRemoteAlerting(some_connectionId, 100, “Trying”)`

OnRemoteAlerting2	SDK raises this event in same time “OnRemoteAlerting”. It has one more argument – ‘lineId’. `OnRemoteAlerting2(int connectionId, int lineId, int responseCode, string reason)`
OnHoldCall	SDK raises this event when local or remote side puts call on hold. `OnHoldCall(int LineId, int isPutOnHold);` Event arguments: `lineId` – id of line, which was held; `isPutOnHold` - is equal 1 when call is held and 0 – otherwise;
OnHoldCall2	SDK raises this event in same time with ‘OnHoldCall’. It has more arguments, which allows to control hold state more precisely. `OnHoldCall2(int lineId, int connectionId, bool isPutOnHold, bool isRemote)` Event arguments: `lineId` – line, which contains this connection; `connectionId` – id of connection, sent this request; `isPutOnHold` – was call put on hold or off hold; `isRemote` – was hold request received from remote side or sent by local side;
OnHoldError	SDK raises this event when detected hold error. `OnHoldError(int lineId, int connectionId, int errCode)` Event arguments: `lineId` – line, which contains this connection; `connectionId` – id of connection, sent this request; `errCode` – detected error code;
OnTextMessageSentStatus	SDK raises this event when received response (expired timeout) on SIP MESSAGE request, sent by method `phone.SendTextMessage`. `OnTextMessageSentStatus(string address, string reason, int bSuccess);` Event arguments: `address` - value received in SIP header 'From'; `reason` – value received in SIP header 'StatusLine' `bSuccess` - is equal 0 if message wasn't send and 1 - otherwise.

OnRecordFinished	SDK raises this event when finished file or buffer recording. Possible cases: - File recording stopped by app manually; - File recording stopped by SDK automatically when call ended; - Buffer recording stopped automatically as buffer is full; `OnRecordFinished(string msg);` Event arguments: `msg` - typical equal to '`Record finished. connId: x`'.
OnRecordFinished2	SDK raises this event in same time with "OnRecordFinished". It has more arguments which allows to control recording more precisely: `OnRecordFinished2(int connectionId, int lineId, long bufferPtr)` Event arguments: `connectionId` – id of connection, sent this request; `lineId` – line, which contains this connection; `bufferPtr` – pointer to recorded buffer;

OnSubscribeStatus	SDK raises this event when received error response (expired timeout) on SIP SUBSCRIBE request, sent by method `phone.Subscribe*`.
	`OnSubscribeStatus(int subscriptionId, int statusCode, string status)`
	Event arguments:
	`subscriptionId` – id assigned by SDK;
	`statusCode` – value received in SIP header 'StatusLine';
	`reason` – value, received in SIP header 'StatusLine';

OnUnSubscribeStatus	SDK never raise this event. `OnUnSubscribeStatus(int subscriptionId, int statusCode, string statusMsg)`

OnSubscriptionRequest	SDK raises this event when received SIP SUBCRIBE request from remote side.
	`OnSubscriptionRequest(string fromUri, string event);`
	Event arguments:
	`fromUri` - value received in SIP header 'From';
	`event` – value received in SIP header 'Event';
	SDK automatically accepts all received subscriptions.

OnSubscriptionTerminated	SDK raises this event when from remote side received request SIP SUBCRIBE with header 'Expires: 0'.
	`OnSubscriptionTerminated(string fromUri)`
	Event arguments:
	`fromUri` - value received in SIP header 'From';

OnSubscriptionNotify	SDK raises this event when received SIP NOTIFY request from remote side.
	`OnSubscriptionNotify(int subscriptionId, string state, string body)`
	Event arguments:
	`subscriptionId` – id assigned by SDK;
	`state` –contains value of XML tag `<note>`, when it present in received body or empty otherwise;
	`body` – body of received SIP request as string;

OnDetectedAnswerTime	SDK has ability to measure duration of period of speech in sound, received from remote side and generate this event (only once per call).
	Explanation:
	Most humans only talk for about 1.5 seconds where answering machines are usually longer than 3.
	See more here:
	http://blog.tropo.com/2010/12/17/human-vs-answering-machinedetection/
	Syntax:
	`OnDetectedAnswerTime(int timeSpanMs, int connectionId);`
	Event arguments:
	timeSpanMs - duration of speech (loud signal level) in received sound.
	connectionId - id assigned to this connection.
	Usage example (C++):
	`HRESULT CMainDlg::AbtoPhone_OnDetectedAnswerTime(LONG TimeSpanMs, LONG ConnectionId)`
	`{`
	`CString str; str.Format(_T("OnDetectedAnswerTime: '%s' connectionId: %d"), (TimeSpanMs > 3000) ? _T("Machine") : _T("Human") , ConnectionId);`
	`DisplayNotifyMsg(str); return S_OK;`
	`}`

Config methods

Name	Description
Store	Method serializes config properties to ini file. Syntax: `config.Store(string fileName);` **Method arguments:** fileName - full path to file or just file name. See more details about file name usage in explanation of method “Load” below.
Load	Method loads phone instance settings from text (ini) file. Syntax: `int s = config.Load(string fileName);` **Method arguments:** `fileName` - full path to file or just file name. Result - non zero if file was read successfully. **Notes** Existing example application are using config file "`phoneCfg.ini`", which allows to modify SDK settings without changing source code of example app. When passed file name only, SDK tries to find it in folder "`C:\Users\-USER-\AppData\Local\VoIP Video SIP SDK`". When file not found SDK searches it in folder with currently running exe file.
StoreAsStr	Method serializes config properties to string and allows app store it. This way is usable in case when app needs to save SDKs settings in own storage and avoid using ini file. Syntax: `string str = config.StoreAsStr();`
LoadFromStr	Method loads config properties from input string. `int result = config.LoadFromStr(string str);` Method arguments: str - string with serialized values. Used same format as SDK writes in config file.

Config properties

Name	Description
ListenPort	Property allows set local port number, which SDK will use for send/receive SIP requests. SDK manages cases when app created few ‘phone’ instances or started few SDK-based apps and set unique port number for all instances. Default value if 5065. Syntax: `Config.ListenPort = x;`
RtpStartPort	Property set range of UDP ports, which SDK uses for send/receive RTP streams. Each SDK instance use range of ports: `[RtpStartPort ... RtpStartPort+32]`. Default value: 17384. Syntax: `Config.RtpStartPort = x;`

ActivePlaybackDevice	Property select speaker device, which SDK will use for playing local and received sound.
	As value of this property app has to use name of device, returned by `Config.PlaybackDevice[]` or one of predefined names: “`None`” or “`Default`”.
	
	Syntax:
	`config.ActivePlaybackDevice =  x;`
	
	Usage examples:
	Select first found speaker device:
	`cfg.ActivePlaybackDevice = cfg.get_PlaybackDevice(0);`
	
	Disable using speaker device:
	`cfg.ActivePlaybackDevice =”None”;`
	
	Use default speaker device:
	`cfg.ActivePlaybackDevice =”Default”;`

PlaybackDevice	This property returns playback device specified by index. Syntax: `String deviceName = config.get_PlaybackDevice(index);` `Index` - value in range `[0..PlaybackDeviceCount-1]`.
PlaybackDeviceCount	Property returns count of available playback devices. Syntax: `int count = config.PlaybackDeviceCount;`

ActiveRecordDevice	This property sets microphone device that have to be used for recording sound.
	
	Syntax:
	`HRESULT ActiveRecordDevice([in] BSTR DeviceName);`
	`HRESULT ActiveRecordDevice([out,retval] BSTR *DeviceName);`
	
	Method arguments:
	DeviceName - existing device name, received from property `IConfig::RecordDevice`

RecordDevice	This property returns microphone device specified by index. Syntax: HRESULT RecordDevice([in] LONG Index, [out, retval] BSTR *pVal); Method arguments: Index - value on range [0..PlaybackDeviceCount-1] pVal - string, returned device name.
RecordDeviceCount	This read-only property returns count of available microphone devices. Syntax: HRESULT RecordDeviceCount([out, retval] LONG *pVal);
RingerDevice	Property allows select speaker device, which SDK will use for playing ringtone. Property value should be name of device, returned by `Config.PlaybackDevice[]` or “`None`” or “`Default`”. `config.RingerDevice = deviceName;`
RingToneEnabled	Property enables/disables playing ring tone when received incoming call. When incoming call received and this option enabled SDK will play sound from file set via property `Config.RingToneFile`. If ringtone file is not set or doesn't found SDK will play single tone with duration 1 second. `Config.RingToneEnabled = true/false;`
RingToneOnCallWaitingEnabled	Property enables/disables playing ring tone when SDK has established call and new incoming call received. `Config.RingToneOnCallWaitingEnabled = true/false;`
RingToneFile	Property set path to file, which SDK will play when incoming call received. `Config.RingToneFile = fileName;` Note: - By default SDK uses file 'telephone-ring.wav'. - When property set as file name only, SDK will try to find it same folder with application exe file.
RingToneFileOnCallWaiting	Property set path to file, which SDK will play when it has established call and new incoming call received. `Config.RingToneFileOnCallWaiting= fileName;`

ActiveNetworkInterface	Property selects network interface, which SDK has to use for sending receiving SIP/RTP packets. As property value app has use string, returned by method ‘`config.get_NetworkInterface`’ or “Auto”. When app set property value "Auto" - SDK will invoke method 'GetBestRoute' with `RegDomain` or `RegProxy` as argument and resolve source network interface, which can send data to `RegDomain`/`RegProxy`. Syntax: `Config.ActiveNetworkInterface = interfaceName;`

NetworkInterface	Property returns network interface, specified by index.
	`string interfaceName = config.NetworkInterface(int Index);`
	Usage example:
	`int ntCount = phoneCfg.NetworkInterfaceCount;`
	`for(int i = 0; i < ntCount; i++)`
	`comboBoxNetwork.Items.Add(phoneCfg.get_NetworkInterface(i));`

NetworkInterfaceCount	Property returns number of available network interfaces. `int ntCount = phoneCfg.NetworkInterfaceCount;`

ActiveVideoDevice	This property sets web camera that will be used for capture video.
	Syntax:
	HRESULT ActiveVideoDevice([in] BSTR DeviceName);
	HRESULT ActiveVideoDevice([out,retval] BSTR *DeviceName);

VideoDevice	This property returns web camera specified by index. Syntax: `HRESULT VideoDevice([in] LONG Index, [out, retval] BSTR *Name);` Method arguments: Index - value on range [0..VideoDeviceCount-1] Name - string, returned video device name.
VideoDeviceCount	This read-only property returns count of available video devices. Syntax: HRESULT VideoDeviceCount([out, retval] LONG *pVal);
SetCodecOrder	Method allows change codecs order. Syntax: `HRESULT SetCodecOrder([in] BSTR CodecksListAsStr, [in] LONG SelectedCount);` Method arguments: CodecksListAsStr - codec’s order separated by '|' char. Example: "G.711 mulaw|G.711 a-law|Speex NB 8,000bps |”. SelectedCount - this argument is obsolete and doesn't used.
CodecCount	This property returns number of loaded codec’s.

	Syntax: `HRESULT CodecCount([out, retval] LONG *Count);`

CodecName	This property returns codec name specified by index.
	Syntax:
	HRESULT CodecName([in] LONG Index, [out, retval] BSTR *Name);
	Method arguments:
	Index - value on range [0..CodecCount-1] Name - string, returned codec name.
	Usage example:
	`int codecCount = phoneCfg.CodecCount;`
	`for(int i = 0; i < codecCount; ++i)`
	`{`
	`   //Add list item:`
	`   phoneCfg.get_CodecName(i)//name of codec item`
	`   phoneCfg.get_CodecSelected(i)//is codec selected`
	`}`

CodecSelected	This property returns non zero when codec specified by index is selected, and zero - in other case. Syntax: `HRESULT CodecSelected([in] LONG Index, [out, retval] LONG *Selected);` Method arguments: Index - value on range [0..CodecCount-1] Selected - non zero when codec is selected, and zero - in other case.

EchoCancelationEnabled	This property enables/disables echo cancelation.
	Syntax:
	`HRESULT EchoCancelationEnabled([in] LONG Enabled);`
	`HRESULT EchoCancelationEnabled([out,retval] LONG *Enabled);`

SilenceDetectionEnabled	This property is obsolete and does nothing.

AutoGainControlEnabled	This property enables/disables auto gain control.
	Syntax:
	`HRESULT AutoGainControlEnabled([in] LONG Enabled);`
	`HRESULT AutoGainControlEnabled([out,retval] LONG *Enabled);`

NoiseReductionEnabled	This property enables/disables noise reduction. Syntax: `HRESULT NoiseReductionEnabled([in] LONG Enabled);`

LicenseKey
LicenseUserId
	Properties set trial/full license credentials, required to initialize SDK instance.
	To prevent serialize these values in cofig file app can see
	Syntax:
	`HRESULT LicenseKey([in] BSTR Key);`
	`HRESULT LicenseKey([out,retval] BSTR *Key);`
	`HRESULT LicenseUserId([in] BSTR UserId);`
	`HRESULT LicenseUserId([out,retval] BSTR *UserId);`

ProxyDomain	Property sets/gets outbound proxy. It’s required in case when IP address of registration domain can’t be resolved via DNS and SDK requires hint, where to send SIP requests. Syntax: HRESULT ProxyDomain([in] BSTR Domain); HRESULT ProxyDomain([out,retval] BSTR *Domain);

ProxyPass	This property sets/gets outbound proxy password.
	Syntax:
	`HRESULT ProxyPass([in] BSTR Pass);`
	`HRESULT ProxyPass([out,retval] BSTR *Pass);`

ProxyUser	This property sets/gets outbound proxy username.
	Syntax:
	`HRESULT ProxyUser([in] BSTR User);`
	`HRESULT ProxyUser([out,retval] BSTR *User);`

RegDomain	Property sets/gets registrar server, eg. 'callcentric.com', 'iptel.org' etc.
	When remote SIP server uses different port than default one ‘5060’ append thus port number to domain name. Example: `altalk.abtollc.com:5088`
	Syntax:
	`HRESULT RegDomain([in] BSTR Domain);`
	`HRESULT RegDomain([out,retval] BSTR *Domain);`

RegUser	Property sets/gets registration user name (extension).
	Syntax:
	`HRESULT RegUser([in] BSTR User);`
	`HRESULT RegUser([out,retval] BSTR *User);`

RegAuthId	Property sets/gets authentication id.
	Use it, when registrar needs other id that differs from extension number.
	Syntax:
	`HRESULT RegAuthId([in] BSTR AuthId);`
	`HRESULT RegAuthId([out,retval] BSTR *AuthId);`

RegExpire	Property sets/gets expiration timeout (seconds, eg. '300').
	Syntax:
	`HRESULT RegExpire([in] LONG Expire);`
	`HRESULT RegExpire([out,retval] LONG *Expire);`
	
	Note:
	When this value is equal to 0, SDK won't send REGISTER message, but 'RegDomain' and 'RegUser’ will be used for outgoing calls.

RegPass	This property sets/gets registration password.
	Syntax:
	`HRESULT RegPass([in] BSTR Pass);`
	`HRESULT RegPass([out,retval] BSTR *Pass);`

RegRealm	This property sets/gets realm which will be associated with username and password credentials.
	
	Syntax:
	`HRESULT RegRealm([in] BSTR Pass);`
	`HRESULT RegRealm([out,retval] BSTR *Pass);`

CallerId	Property configures caller id (display name).
	When caller id is not empty, SDK uses it in SIP headers like: "caller_id" <extension@domain>.
	Syntax:
	`HRESULT CallerId([in] BSTR Id);`
	`HRESULT CallerId([out,retval] BSTR *Id);`

SetRegisterBody	Method allows set string with body and content type, which SDK has to add to REGISTER request. `Config.SetRegisterBody(string body, string contentType);`
StunServer	Property set STUN server, which SDK will use to resolve public IP:port. `Config.StunServer = “stun.l.google.com:19302”;`
OverridePublicIP	Property allows set `IP:port`, which SDK will use in ‘Contact’ header and also in SDP body. `Config.OverridePublicIP = 10.0.0.10:2222;`

AllowedRemoteIP	Property enables filtering incoming SIP requests.
	SDK will handle only requests, received from specified `IP:port` and silently skip all others.
	`HRESULT AllowedRemoteIP([in] BSTR srcAddr);`
	`HRESULT AllowedRemoteIP([out,retval] BSTR *srcAddr);`

OverrideRtpDest	Property enables sending local RTP to specified IP address.
	In this mode SDK ignores IP address, received in SDP body from remote side.
	`HRESULT OverrideRtpDest([in] BSTR dest);`
	`HRESULT OverrideRtpDest([out,retval] BSTR *dest);`

ReplyRtpBackToSenderEnabled	Property enables sending local RTP to IP:port from which was received remote one.
	This mode is usable when app is working on computer with public IP and established call with client under symmetric NAT.
	`HRESULT ReplyRtpBackToSenderEnabled([in] VARIANT_BOOL Enabled);`
	`HRESULT ReplyRtpBackToSenderEnabled([out,retval] VARIANT_BOOL *Enabled);`

UserAgent	This property sets/gets string that is placed in SIP message INVITE, header - "UserAgent". Syntax: HRESULT UserAgent([in] BSTR AgentName);
SamplesPerSecond	Property set sample rate, which SDK uses internally in media graphu. When sample rate of audio devices, codecs, files is different SDK resamples automatically. Default value is 44100 samples per second. Syntax: HRESULT SamplesPerSecond([in] LONG Samples); HRESULT SamplesPerSecond([out,retval] LONG *Samples);

DialToneEnabled	This property enables/disables playing internal dial tones (when received event OnRemoteAlerting(n, 180, "Ringing")).
	
	Syntax:
	HRESULT DialToneEnabled([in] LONG Enabled);
	HRESULT DialToneEnabled([out,retval] LONG *Enabled);

VolumeUpdateSubscribed	This property sets/gets subscription state for event _IAbtoPhoneEvents::OnVolumeUpdated. Subscription is needed because of this event is generated fairly often, and not all clients wants to handle it. Look also _IAbtoPhoneEvents::OnVolumeUpdated. Syntax: HRESULT VolumeUpdateSubscribed([in] LONG Enabled); HRESULT VolumeUpdateSubscribed([out,retval] LONG *Enabled);

LogLevel	This property sets/gets log level.
	SDK creates log file "SIPVoipSDKLog.txt" in folder where registered "SIPVoIPSDK.dll".
	Possible log levels are: eLogNone, eLogCritical, eLogError, eLogWarning, eLogInfo, eLogDebug.
	
	Syntax:
	`HRESULT LogLevel([in] LogLevelType level);`
	`HRESULT LogLevel([out,retval] LogLevelType *level);`

LogPath	Property allows set path, where SDK creates log file SIPVoipSDKLog.txt.
	By default SDK creates log file in: "`C:\Users\-USER-\AppData\Local\VoIP Video SIP SDK`".
	`HRESULT LogPath([in] BSTR logFilePath);`
	`HRESULT LogPath([out,retval] BSTR *logFilePath);`

AdditionalDnsServer	This property sets/gets additional dns server string.
	This feature may be need when SDK reports error like "503 No DNS result found".
	Sometimes can help using Google open DNS "8.8.8.8".
	
	Syntax:
	`HRESULT AdditionalDnsServer([in] BSTR dnsServerIPAddress);`
	`HRESULT AdditionalDnsServer([out,retval] BSTR *dnsServerIPAddress);`

AdditionalSDPContent	Method allows configure own SDP attributes, which SDK will add to SDP body of outgoing INVITE request. Usage example: `Config.AdditionalSDPContent = "attrib1=value1| atrib2=value2|… ";`

MP3RecordingEnabled	Property enables encoding recorded sound to mp3 format,.
	This option has no effect for buffer recording `AbtoPhone::StartRecordingBuffer`;
	Syntax:
	`HRESULT MP3RecordingEnabled([in] LONG Enabled);`
	`HRESULT MP3RecordingEnabled([out,retval] LONG *Enabled);`

LocalAudioEnabled	Property enables ability to work without audio devices. When this option enabled SDK doesn't check are audio devices connected and also doesn't play any sounds.
	Syntax:
	`HRESULT LocalAudioEnabled([in] LONG Enabled);`
	`HRESULT LocalAudioEnabled([out,retval] LONG *Enabled);`

LocalTonesEnabled	This property disables/enables playing DTMF tones on local side when they are sending via method IAbtoPhone::SendTone.
	
	Syntax:
	`HRESULT LocalTonesEnabled([in] LONG Enabled);`
	`HRESULT LocalTonesEnabled([out,retval] LONG* Enabled);`

LocalVideoWindow	Property sets/gets window handle (HWND), where SDK will render image, captured from local camera.
	Syntax:
	`HRESULT LocalVideoWindow([in] LONG hVideoWindow);`
	`HRESULT LocalVideoWindow([out,retval] LONG* hVideoWindow);`

RemoteVideoWindow	Property sets/gets window handle, where SDK will render image, received from remote side.
	
	Syntax:
	`HRESULT RemoteVideoWindow([in] LONG hVideoWindow);`
	`HRESULT RemoteVideoWindow([out,retval] LONG* hVideoWindow);`

VideoCallEnabled	This property enables/disables all video features.
	When this option enabled SDK can send/receive video data.
	
	Syntax:
	`HRESULT VideoCallEnabled([in] LONG Enabled);`
	`HRESULT VideoCallEnabled([out,retval] LONG *Enabled);`

CallInviteTimeout	Property defines how long SDK will wait answer from remote side after sending INVITE request.
	
	Syntax:
	`HRESULT CallInviteTimeout([in] LONG timeoutSeconds);`
	`HRESULT CallInviteTimeout([out,retval] LONG *timeoutSeconds);`
	
	Note:
	timeoutSeconds - value in seconds. Default value is equal to 40 (second).

EncryptedCallEnabled	Property enables SRTP for sending audio/video.
	Call won't be established when this option is enabled, but remote side doesn't support SRTP.
	
	Syntax:
	`HRESULT EncryptedCallEnabled([in] LONG Enabled);`
	`HRESULT EncryptedCallEnabled([out,retval] LONG *Enabled);`

KeepAliveTimeSIP	Property enables sending short packets via SIP signaling transport to remote server, which has prevent closing UDP ports between app and server.
	By default this option is disabled (equal to 0).
	Recommended value is 30 (send each 30seconds).
	
	Syntax:
	`HRESULT KeepAliveTimeSIP([in] LONG seconds);`
	`HRESULT KeepAliveTimeSIP([out,retval] LONG *seconds);`

KeepAliveTimeRTP	Property enables sending short packets via RTP transport which has prevent closing UDP ports between app and server. Option is usable in case when app disabled sending own video/audio, but requires to receive remote.

	Syntax:
	`HRESULT KeepAliveTimeRTP([in] LONG seconds);`
	`HRESULT KeepAliveTimeRTP([out,retval] LONG *seconds);`

ICEEnabled	This property allows enable/disable ICE (Interactive Connectivity Establishment) feature.
	Syntax:
	`HRESULT ICEEnabled([in] LONG Enabled);`
	`HRESULT ICEEnabled([out,retval] LONG *Enabled);`
	Note:
	When this option is enabled SDK includes in SDP section of SIP INVITE message list of candidates that can be used for RTP connection.
	Also has to be set STUN setting. Recommended STUN value is `stun.l.google.com:19302`.

DtmfAsSipInfoEnabled	Property enables sending DTMF as SIP INFO request.
	This property is used only by one method: `phone.SendTone(string t)`
	Syntax:
	`HRESULT DtmfAsSipInfoEnabled([in] LONG Enabled);`
	`HRESULT DtmfAsSipInfoEnabled([out,retval] LONG *Enabled);`

VideoFrameWidth
VideoFrameHeight
	These properties allows set resolution of frame, which SDK will capture from selected local video device. When requested resolution is not supported SDK will scale it.
	Default resolution is CIF 352x288.
	Syntax:
	`HRESULT VideoFrameWidth([in] LONG Width);`
	`HRESULT VideoFrameWidth([out,retval] LONG *Width);`
	`HRESULT VideoFrameHeight([in] LONG Width);`
	`HRESULT VideoFrameHeight([out,retval] LONG *Width);`

VideoBitRate	Property allows set video codec bitrate which allows to set higher compression and reduce amount of data, sent to network, or set higher quality.
	Default value is 0 – which means default codec settings.
	`HRESULT VideoBitRate([in] LONG Rate);`
	`HRESULT VideoBitRate([out,retval] LONG *Rate);`

VideoFrameRate	Property set how many video frames per second SDK has encode and send to remote side.
	Default value is 15.
	`HRESULT VideoFrameRate([in] LONG Rate);`
	`HRESULT VideoFrameRate([out,retval] LONG *Rate);`

VideoAutoSendEnabled	Property enables sending own video to remote side when call established.
	It’s usable when is required to implement “doorphone” mode – app is able to see remote video, but doesn’t send own, local one.
	`HRESULT VideoAutoSendEnabled([in] LONG Enabled);`
	`HRESULT VideoAutoSendEnabled([out,retval] LONG *Enabled);`

VideoResizeToWindow	Property allows disable scaling of received video frame. For example:

	when received video frame with resolution 640x480, but local window, where SDK has to render this frame, has different size SDK will scale received frame to fit window's rect.
	`HRESULT VideoResizeToWindow([in] LONG Enabled);`
	`HRESULT VideoResizeToWindow([out,retval] LONG *Enabled);`

SDPInRingingMsgEnabled	Property allows to implement receiving early media.
	Example of use case:
	- Received incoming call
	- SDK answer "180 Ringing + SDP", which allows to receive audio/video from remote side.
	Syntax:
	`HRESULT SDPInRingingMsgEnabled([in] LONG Enabled);`
	`HRESULT SDPInRingingMsgEnabled([out,retval] LONG *Enabled);`

AutoAnswerEnabled	Property enabled automatic answers all incoming calls.
	Syntax:
	`HRESULT AutoAnswerEnabled([in] LONG Enabled);`
	`HRESULT AutoAnswerEnabled([out,retval] LONG *Enabled);`

MixerFilePlayerEnabled	Property switches file playing modes.
	When this option enabled (default state) - SDK plays files via internal mixer.
	This allows send to remote side mixed sound from microphone, file and tones.
	Disadvantage of this method - only single line can play file in same time.
	
	When this option disabled - SDK plays files via line-dependent player.
	This allows play different files in different lines in same time, but local sound from microphone can't be sent during playing.
	
	Syntax:
	`HRESULT MixerFilePlayerEnabled([in] LONG Enabled);`
	`HRESULT MixerFilePlayerEnabled([out,retval] LONG *Enabled);`

SignallingTransport	Property set signaling transport for SIP protocol.
	Allowed values are: `eTransportUDP, eTransportTCP, eTransportTLSv1`
	Default value is `eTransportUDP`.
	Syntax:
	`HRESULT SignallingTransport([in] TransportType transp);`
	`HRESULT SignallingTransport([out,retval] TransportType *transp);`

TlsVerifyServerEnabled	Property enables verify domain of remote SIP server before establish TLS connection.
	It compares destination domain with CN value, received in certificate from server.
	By default it's disabled.
	
	When app enables this mode it has provide CA cert file – certificate of CA, which signed server's certificate.
	Copy this file to folder with SDK dll and rename to "`root_cert_<name>.crt`".
	Syntax:
	`HRESULT TlsVerifyServerEnabled([in] VARIANT_BOOL Enabled);`
	`HRESULT TlsVerifyServerEnabled([out,retval] VARIANT_BOOL *Enabled);`

TlsForceSipScheme	Property enables using 'sip' scheme even when TLS transport is selected.
	Disabled by default;
	`HRESULT TlsForceSipScheme([in] VARIANT_BOOL Enabled);`
	`HRESULT TlsForceSipScheme([out,retval] VARIANT_BOOL *Enabled);`

LoopbackNetworkInterfaceEnabled	Property enables sending requests via loopback network interface (127.0.0.1).
	By default property is disabled.
	Syntax:
	`HRESULT LoopbackNetworkInterfaceEnabled([in] LONG Enabled);`
	`HRESULT LoopbackNetworkInterfaceEnabled([out,retval] LONG *Enabled);`

ExSipAccount_Count	Property returns number of added SIP accounts.
	Syntax:
	`ExSipAccount_Count([out, retval] LONG *pVal)`
	Example
	`int c = phoneCfg.ExSipAccount_Count();`

ExSipAccount_Get	Method returns SIP account specified by index.
	Syntax:
	`HRESULT ExSipAccount_Get([in] LONG ExAccountIdx,`
	`[out] BSTR* Domain, [out] BSTR* User, [out] BSTR* Pass, [out] BSTR* AuthId,`
	`[out] BSTR* DisplName, [out] LONG* Expire, [out] BOOL* bEnable)`
	Method arguments:
	ExAccountIdx - value on range [0..ExSipAccount_Count-1].
	Domain - string, registrar server. Similar with value managed via property 'RegDomain'.
	User - string, registration user name. Similar with value managed via property 'RegUser'.
	Pass - string, registration password. Similar with value managed via property 'RegPass'.
	AuthId - string, authentification id. Similar with value managed via property 'RegAuthId'.
	DisplName - string, display name. Similar with value managed via property 'CallerId'.
	Expire - integer, registration expiration timeout. Similar with value managed via property 'RegExpire'. bEnable - bool, is this account enabled.

ExSipAccount_GetDefaultIdx	Method returns index of default account (used for outgoing calls). Syntax: `HRESULT ExSipAccount_GetDefaultIdx([out, retval] LONG* ExAccountIdx)`

ExSipAccount_SetDefaultIdx	Method set default account, which SDK will use for outgoing calls.
	Syntax:
	`HRESULT ExSipAccount_SetDefaultIdx([in] LONG ExAccountIdx)`
	For easier implementation use method 'StartCallAcc', which allows specify account for outgoing call.
	Example:
	`int connId = AbtoPhone.StartCallAcc(number, accountIdx);`

ExSipAccount_ResetAll	Method clears all added accounts. Syntax: `HRESULT ExSipAccount_ResetAll();`

ExSipAccount_Add	Method adds new SIP account.
	Syntax:
	`HRESULT ExSipAccount_Add([in] BSTR Domain, [in] BSTR User, [in] BSTR Pass,`
	`[in] BSTR AuthId, [in] BSTR DisplName, [in] LONG Expire,`
	`[in] BOOL bEnable, [in] BOOL bDefault);`

	Method arguments:
	Same with method "ExSipAccount_Get' bDefault - has to be set as 'true', when default account is adding.
	
	Usage example:
	`//Get config interface`
	`CConfig phoneCfg = AbtoPhone.Config;`
	
	`//Reset existing accounts`
	`phoneCfg.ExSipAccount_ResetAll();`
	
	`//Add accounts`
	`phoneCfg.`**`ExSipAccount_Add`**`(domain1, user1, pass1, "", "", 300, true, 0);`
	`phoneCfg.`**`ExSipAccount_Add`**`(domain2, user2, pass2, "", "", 300, true, 0);`
	
	`//Set default account (0-based index) which SDK will use for outgoing calls`
	`phoneCfg.`**`ExSipAccount_SetDefaultIdx`**`(1);`
	
	`//Apply changes`
	`AbtoPhone.ApplyConfig();`

ExSipAccount_Remove	Method removes account by index. `HRESULT ExSipAccount_Remove([in] LONG ExAccountIdx);`

ComfortNoiseOnMutedMicEnabled	Property enables generating and sending comfort noise to remote side of call, when missed signal from microphone.
	Syntax:
	`HRESULT ComfortNoiseOnMutedMicEnabled([in] LONG Enabled);`
	`HRESULT ComfortNoiseOnMutedMicEnabled([out,retval] LONG *Enabled);`

SendRingingMsgEnabled	Option enables sending “180 Ringing” request when received incoming call.
	Enabled by default.
	Syntax:
	`HRESULT SendRingingMsgEnabled([in] LONG Enabled);`
	`HRESULT SendRingingMsgEnabled([out,retval] LONG *Enabled);`

AudioQosDscpVal
VideoQosDscpVal
	Properties allows set value of DSCP field in IP header of each RTP packet.
	This feature requires to start app with admin permissions. See more:
	https://docs.microsoft.com/en-us/windows/desktop/api/qos2/nf-qos2-qoscreatehandle
	
	Syntax:
	`HRESULT AudioQosDscpVal([in] LONG val);`
	`HRESULT AudioQosDscpVal([out,retval] LONG *val);`
	`HRESULT VideoQosDscpVal([in] LONG val);`
	`HRESULT VideoQosDscpVal([out,retval] LONG *val);`
	
	Usage example:
	`phone.Config.AudioQosDscpVal = 46;`
	`phone.Config.VideoQosDscpVal = 46;`

AudioSSRCVal
VideoSSRCVal
	Properties allows override SSRC value of audio/video RTP streams.
	Syntax:
	`HRESULT AudioSSRCVal([in] LONG val);`
	`HRESULT AudioSSRCVal([out,retval] LONG *val);`
	`HRESULT VideoSSRCVal([in] LONG val);`
	`HRESULT VideoSSRCVal([out,retval] LONG *val);`

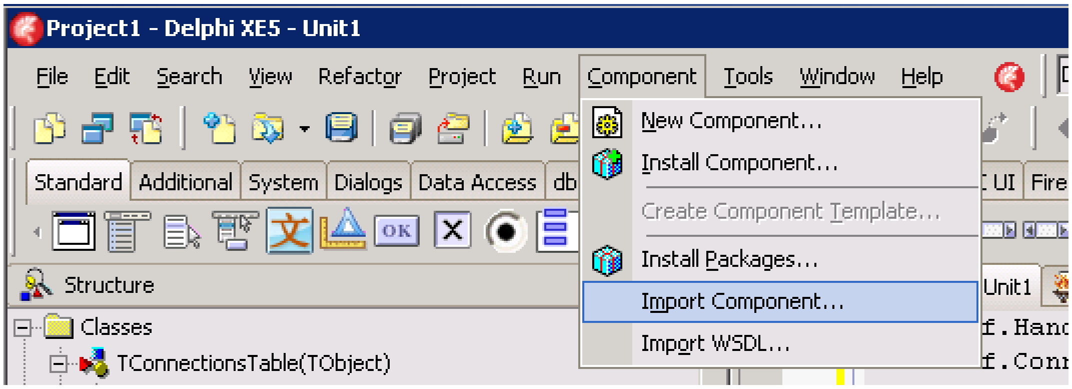
DmpCreateEnabled	SDK has ability to intercept unhandled exceptions and create mini dump files.
	Possible values:
	1 - Display message box with exception and create dmp file. Default value;
	0 - Silently create dmp file.
	-1 - Don't display message boxex nor create dmp files;
	
	Syntax:
	`HRESULT DmpCreateEnabled([in] LONG Enabled);`
	`HRESULT DmpCreateEnabled([out,retval] LONG *Enabled);`

AppendExtToRecFileNameEnabled	Property enables adding "wav"/"mp3" extension to recording file name.
	It's usable in case when app requires to use own nonstandard extension, which already included in 'FilePath' argument of method `phoneStartRecording`.
	Syntax:
	`HRESULT AppendExtToRecFileNameEnabled([in] VARIANT_BOOL Enabled);`
	`HRESULT AppendExtToRecFileNameEnabled([out,retval] VARIANT_BOOL *Enabled);`

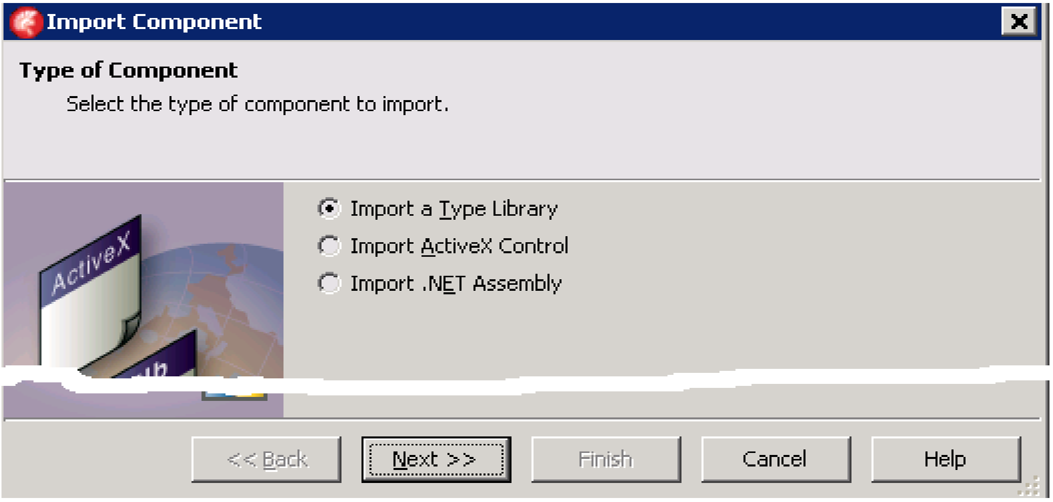
How to add VoIP SIP SDK to the application and make calls

Tutorial for Delphi

Go to menu: Project->Import Component


Project1 - Delphi XE5 - Unit1
File Edit Search View Refactor Project Run Component Tools Window Help
Standard Additional System Dialogs Data Access db
Structure
Classes
TConnectionsTable(TObject)
New Component...
Install Component...
Create Component Template...
Install Packages...
Import Component...
Import WSDL...
Unit1

Import type library



Type of Component

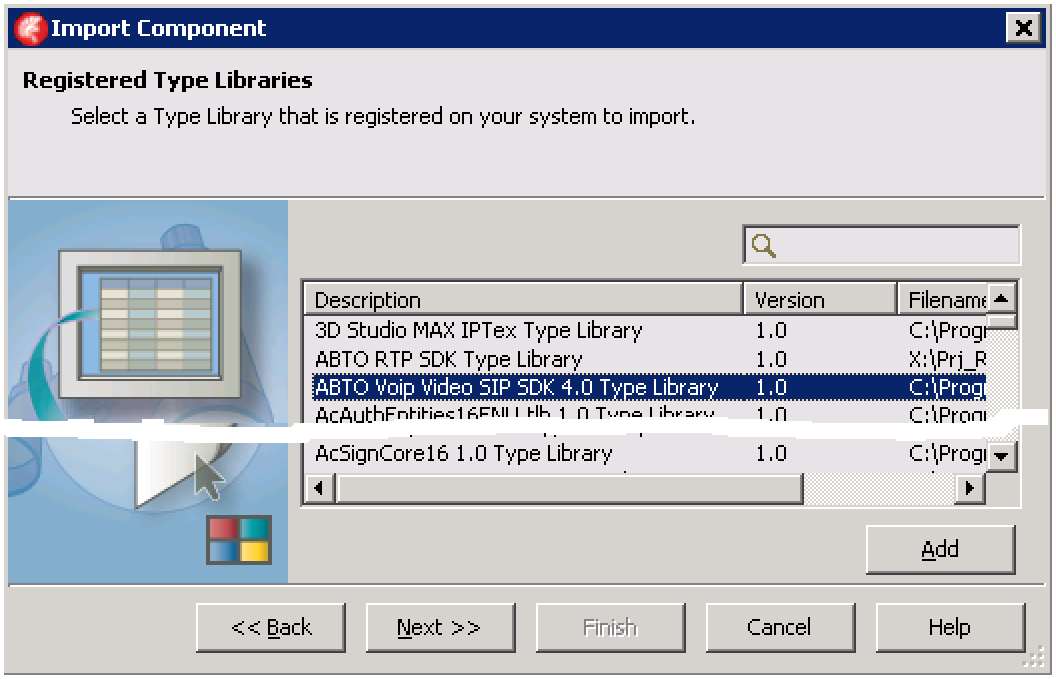
Select the type of component to import.

● Import a Type Library

○ Import ActiveX Control

○ Import .NET Assembly

Locate "ABTO SIP SDK 4.0 Type Library"



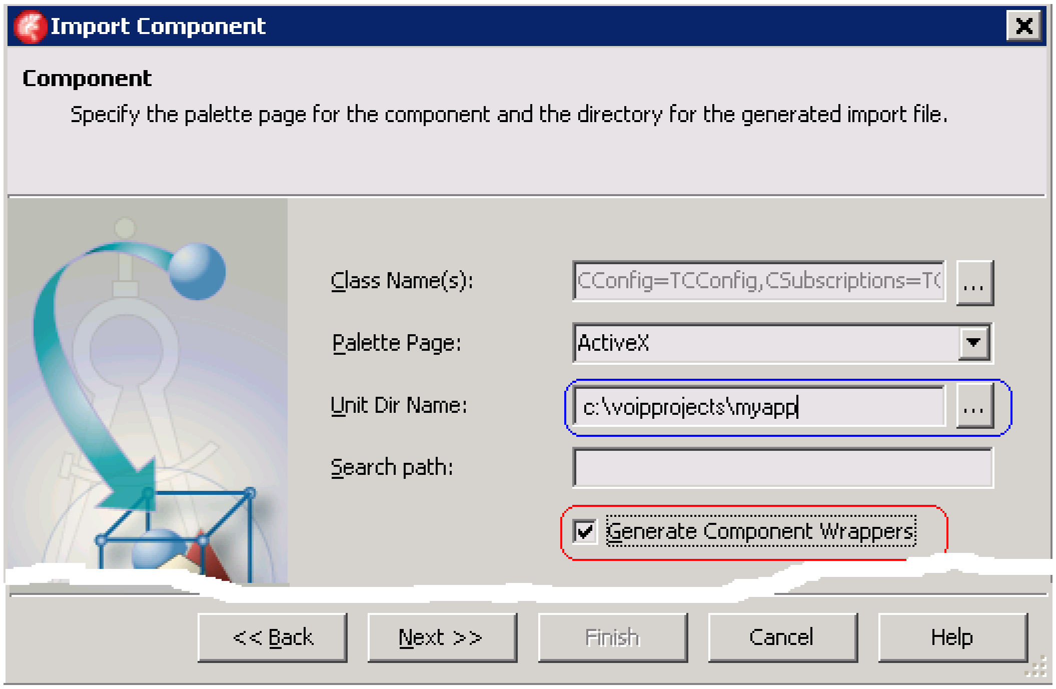
Import Component

Registered Type Libraries

Select a Type Library that is registered on your system to import.

Description	Version	Filename
3D Studio MAX IPTex Type Library	1.0	C:\Progr
ABTO RTP SDK Type Library	1.0	X:\Prj_R
ABTO Voip Video SIP SDK 4.0 Type Library	1.0	C:\Prog
AcAuthEntities16ENU.tlb 1.0 Type Library	1.0	C:\Progr
AcSignCore16 1.0 Type Library	1.0	C:\Progr

Enter path to project, where to generate wrappers



Import Component

Component

Specify the palette page for the component and the directory for the generated import file.

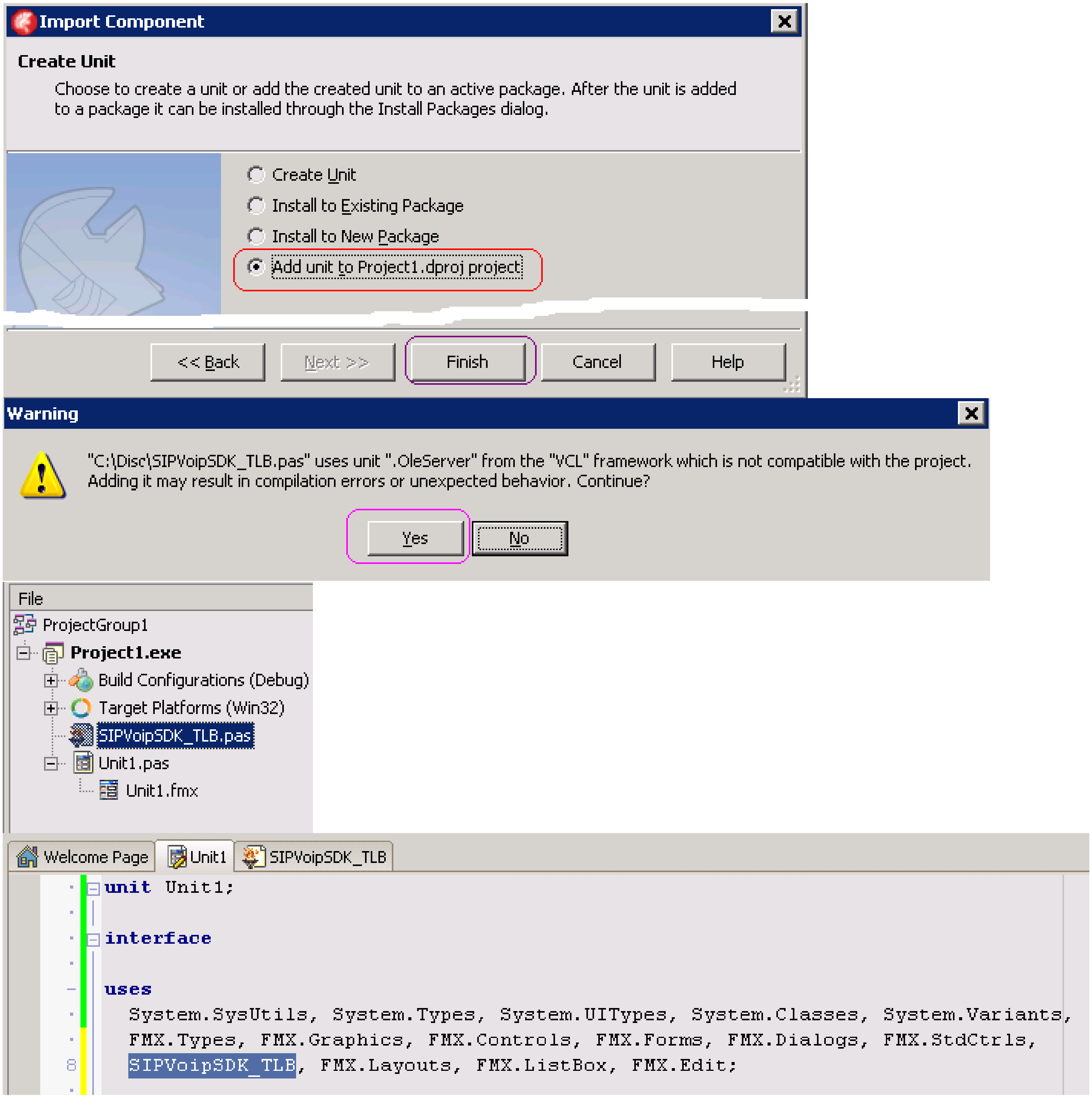
Class Name(s): CConfig=TCConfig,CSubscriptions=T(

Palette Page: ActiveX

Unit Dir Name: c:\voipprojects\myapp

Search path:

☑ Generate Component Wrappers



Import Component

Create Unit

Choose to create a unit or add the created unit to an active package. After the unit is added to a package it can be installed through the Install Packages dialog.

○ Create Unit
○ Install to Existing Package
○ Install to New Package
● Add unit to Project1.dproj project

Warning

"C:\Disc\SIPVoipSDK_TLB.pas" uses unit ".OleServer" from the "VCL" framework which is not compatible with the project. Adding it may result in compilation errors or unexpected behavior. Continue?

```
unit Unit1;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Graphics, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.StdCtrls,
  SIPVoipSDK_TLB, FMX.Layouts, FMX.ListBox, FMX.Edit;
```

In edit box "Unit dir name:" type full path to current project folder.
Press "Create unit"
Will be generated new unit "SIPVoipSDK_TLB" in file SIPVoipSDK_TLB.pas.

Go to the source code and add declarations

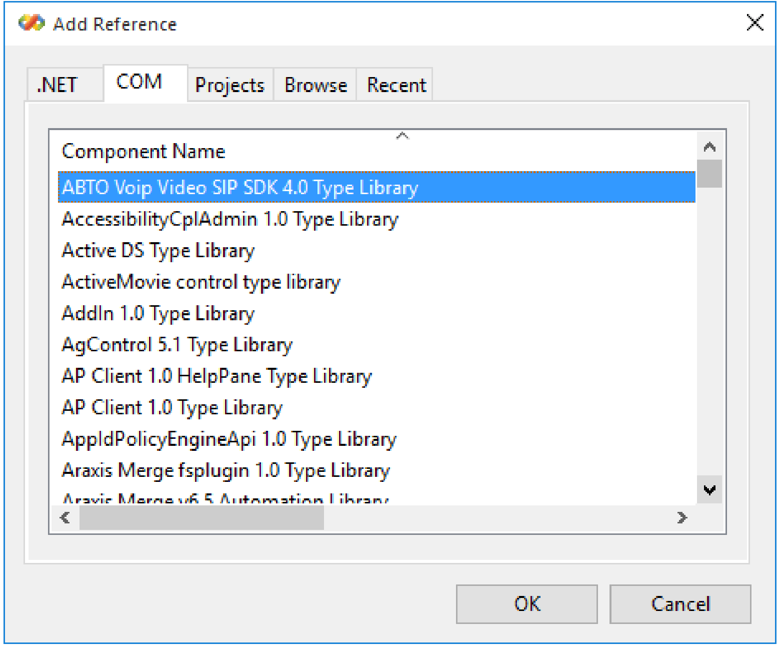
```
private AbtoPhone: TCAbtoPhone;

procedure TForm1.FormCreate(Sender: TObject); var
   phoneCfg : Variant;
begin
   AbtoPhone := SIPVoipSDK_TLB.TCAbtoPhone.Create(Self);
   phoneCfg := AbtoPhone.Config;

   {Note: Uncomment following lines if needed}
   //Log level
   //phoneCfg.LogLevel := LogLevelType.eLogDebug;//eLogError

   //Specify Licensy key
   phoneCfg.LicenseUserId := '...';
   phoneCfg.LicenseKey := '...';

   AbtoPhone.ApplyConfig;//Apply modified config

   try
      AbtoPhone.Initialize; //Initialize component
   except    end;
end;
```

Run application.

Tutorial for Visual Studio (C#)

Open C# project, Add reference to SIP SDK:

Right click on "References" in Solution Explorer, choose "Add reference", Switch to "COM" tab, Locate "ABTO SIP SDK 4.0 Type Library"



Component Name
ABTO Voip Video SIP SDK 4.0 Type Library
AccessibilityCplAdmin 1.0 Type Library
Active DS Type Library
ActiveMovie control type library
AddIn 1.0 Type Library
AgControl 5.1 Type Library
AP Client 1.0 HelpPane Type Library
AP Client 1.0 Type Library
AppIdPolicyEngineApi 1.0 Type Library
Araxis Merge fsplugin 1.0 Type Library
Araxis Merge v6.5 Automation Library

Press OK.

Add SDK initialization code.

Add using directive (in bottom of existing "using..." declaration)

```
using SIPVoipSDK;
```

Declare and initialize phone instance variable:

```
private CAbtoPhone AbtoPhone = new CAbtoPhone();
```

Add code to initialize 'phone' instance

```
private void SIPPhone_Load(object sender, EventArgs e)
{
   //Get config CConfig
   CConfig phoneCfg = AbtoPhone.Config;

   //Specify Licensy key
   phoneCfg.LicenseUserId = "...";
   phoneCfg.LicenseKey = "...";

   //Log level
   phoneCfg.LogLevel = LogLevelType.eLogDebug;//eLogError

   //Apply modified config
   AbtoPhone.ApplyConfig();

   try {
      AbtoPhone.Initialize();
   }
   catch(Exception exc)
   {
      MessageBox.Show(exc.Message, "Error", MessageBoxButtons.OK);
   }
}
```

Add code to start a call

```
private void CallButton_Click(object sender, EventArgs e)
{
   int connectionId = AbtoPhone.StartCall2(AddressBox.Text);

}
```

FAQ. PROBLEMS AND SOLUTIONS

How to copy SDK on another computer?

Copy following files from SDK folder "binary" to clients (users) computer:

```
avcodec-53.dll
avutil-51.dll
swscale-2.dll
codec_g729.dll
codec_gsm.dll
codec_pcmapcmu.dll
codec_tones.dll
Interop.SIPVoipSDK.dll
Microsoft.VC80.CRT.manifest
msvcp80.dll
msvcr80.dll
SIPVoipSDK.dll
```

Register SDK ActiveX

Start `cmd.exe` with administrator permissions;

Go to folder, where were copied files on previous stage

Example: `cd "c:\Program Files\SomeVoipApp"`

Type command:

```
regsvr32 SIPVoIPSDK.dll
```

How to reject second and next calls when there is call established

Requested feature can be implemented in following way:

```
int gEstablishedConnectionId = 0;

void AbtoPhone_OnIncomingCall2(string From, string To, int connId, int lineId)
{
  if (gEstablishedConnectionId == 0)
  {
    gEstablishedConnectionId = connectionId;
    Phone.AnswerCallLine(lineId);
  }
  else
  {
    Phone.RejectCallLine(lineId);
  }
  ...
}

private void AbtoPhone_OnClearedConnection(int connectionId, int lineId)
{
  if (gEstablishedConnectionId == connectionId) gEstablishedConnectionId = 0;
}
```

Are there any Fax detection options?

SDK is able to detect FAX signals: “FAX 2100” and “FAX 1100”.
To enable detection set option on initialization stage:

```
Phone.Config.TonesTypesToDetect = (int)ToneType.eToneMF;
```

When detected fax signal SDK will raise event:

```
OnToneDetected(ToneType.eToneMF, “FAX 2100",  connectionId, lineId)
```

Shutting down

"Do I need to unregister? Is there a sequence I should be using when I want to shut down the softphone? Currently I am not doing anything".
Response
In general, you don't have to do anything specific, unless there are active calls. In such case, you should "Hang up" first to shutdown correctly. SDK will unregister automatically before exit or new registration.

How to update current SDK version?

Download "SIPVoipSDK.dll_xxx" from provided link
Copy to folder where it's registered/installed.
Run app.

How to verify version of current SDK and path where it's registered

```
string version = phone.RetrieveVersion();//version of loaded SDK dll
string path    = phone.SDKPath();         //path, from which was loaded SDK dll
```

How to get CallerID on incoming call?

When received incoming call SDK raises event:

```
OnIncomingCall2(string AddrFrom, string AddrTo, int connectionId, int lineId)
```

Where argument 'AddrFrom' - contains value received in SIP header 'From'.
It has following format: "[CallerId] <sip:[Extension]@[domain]>".
Example:

```
"NewUser" <sip:1000@192.168.0.178>
"sip:1000@192.168.0.178"
```

What is the config file phoneCfg.ini?

SDK has ability to serialize IConfig's properties to the text file via methods Config.Load, Config.Store.
Default name of config file is "phoneCfg.ini" and it can be used from all SDK examples applications.

App can use own file name or doesn't use config file.

I just looked for the threading model of the SIPVoipSDK.dll

We suggest to create SDKs 'phone' instance and use it from same GUI thread.
Its internal implementation is based on queue of commands.
When app invokes some SDKs method it just posts command to internal queue, which is handled by background thread.
Methods 'Initialize' and 'ApplyConfig' are synchronous - app has to wait they end.
By default SDK instance raises events in same thread where it's created.
Method 'phone.InitializeEx(0)' - allows initialize SDK in way when it raises event from other thread.

I don't use video, are there codecs I can drop.

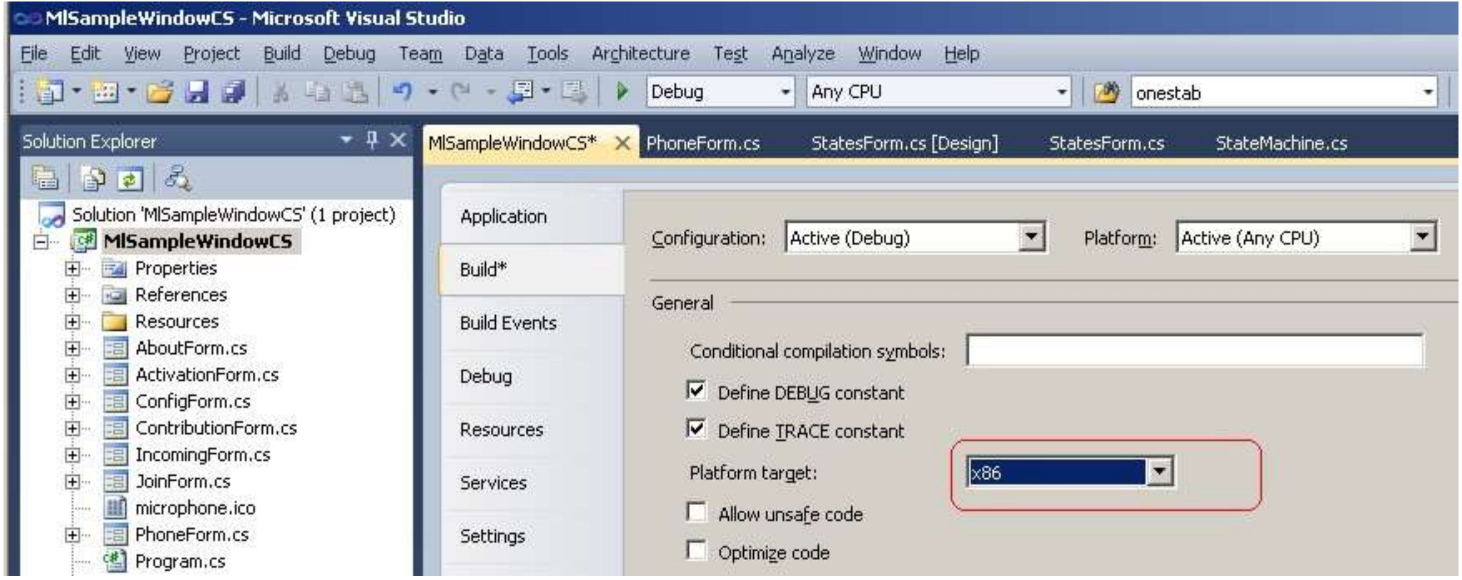
We can provide specific SDK build without video call features.
It will not require: avcodec-53.dll, avutil-51.dll, swscale-2.dll

Is lame.exe required for the SDK?

SDK invokes lame.exe when is required to convert *.mp3 -> *.wav in method `phone.PlayFile`
If playing mp3 is not required - you can remove that file.

C# app doesn't work on x64 machine

When app compiled with setting “`Platform target = AnyCPU`” and uses x86 version of SDK it will work good on x86 machine and will not work on x64.
To fix this issue – use x64 version of SDK or set project setting “`Platform target = x86`” as shown on image below.



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

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

AnswerCall and RejectCall have no parameters

When two calls arrive at the same time on how can I decide which call to Answer/Reject?

Use event:
`OnIncomingCall2(string AddrFrom, string AddrTo, int connectionId, int lineId)`
and remember '`lineId`', in some context variable.

When user decided to answer/reject call invoke:
`phone.AnswerCallLine(lineId);`
or
`phone.RejectCallLine(lineId);`

How to set domain/proxyport number if it differs from 5060

Define it with a colon behind the ip address:
`domain-with-port.proxy.com:`**`4912`**

I don't understand the parameters of SendToneExLine and SendToneExLine2.

SDK supports 3 ways of sending DTMF:
`bSendAudio_InBand` - generate tone sound, mix with microphone and send to remote side
`bSend_RFC4733_OutOfBand` - generate and specific RTP packets
`bSend_SIP_INFO` - send SIP INFO request
Recommended to enable 'InBand'+'RFC4733'

How to specify selected codecs and order:

Use method ‘SetCodecOrder’ and build string argument, which contains list of codecs.
Example:

```
phone.Config.SetCodecOrder("G.729A|G.711 mu-law|G.711 a-law|RFC4733 DTMF tones|H.263-1998|H.264-
BP10|");
```

I want to send sequence of tones to the current connection?

Use code

```
phone.SendToneExLine(0, connId, '#*0', 100, 1,1, 0);
```

SDK will ignore 'lineId' argument (as it's 0) and send sequence of tones to specified connection with duration 100ms, as RTP signaling + in-band sound.

How to record the local recording device, or how to record the mix of both devices.

Audio subsystem inside SDK has mixer, which allows to create conference calls and also mix local and remote sounds.
Method `StartRecordingConnection(connectionId)` records only received sound (before handling it by mixer);
Also `StartRecordingConnection(0)` - records only local microphone (before handling it by mixer);
`StartRecording` - records mixer output, which includes local+remote+file player.

When app started few calls, local user can hear only one of them - call on current line.
User can switch calls, using method `phone.SetCurrentLine(2)` - which causes switching mixer gains.
In this case recorder will capture sound from call on line 2 + local mic.

With AutoAnswer on, our customers wish to have a tone played in the operator's ear before the call is connected.

Prepare sound file with signal, which is required to play when call established, and add code:

```
void AbtoPhone_OnEstablishedCall(string adress, int lineId)
{
   AbtoPhone.BeepFile(“beep.wav”, 80, BeepFileDirection.eLocalOnly);
```

How to unregister account.

Use code

```
Phone.Config.RegExpire=0;
Phone.ApplyConfig();
```

To register back:

```
Phone.Config.RegExpire=300;
Phone.ApplyConfig();
```