Agora Windows SDK Reference

Contents

Required Libraries

- Agora Audio SDK requires Visual C++ 2008 x86 runtime libraries.
- Add the AgoraAudioSDK\include directory to the INCLUDE directories of your project.
- Add the 'AgoraAudioSDK\lib' directory to the LIB directories of your project and make sure mediasdk.lib is linked with your project.
- Copy dlls under AgoraAudioSDK\dll to the directory where your executable file is located.

AgoraAudio Methods

Create Agora Audio Object

AgoraAudio(IAgoraAudioEventHandler* handler);

or

AgoraAudio = createAgoraAudioInstance(EventHandler);

This initializes the AgoraAudioKit class. The EventHandler interface methods are called every three seconds for every caller on the call (channel) to provide call quality information.

initWithQuality function	
Argument	Description
handler	IAgoraAudioEventHandler (See below.)

Join Channel

This method lets users join a channel. Think of that as a chat room, except that it is a multi-party phone call. This method is asynchronous, so it can be called on the main UI thread.

void joinChannel(const char* vendorKey, const char* channelName, const char* info, unsigned int uid)

Name	Description
vendorKey	Account credentials issued by Agora Voice to app developer, i.e., a user license.
channelName	Channel name. Any descriptive name like “game1” or “call2”.
info	Optional. Whatever the additional information the programmer wants to

	add.
uid	Optional. User id. If you do not set one the SDK supplies one.

Leave Channel

Leave channel, meaning hang up or exit call.

void release()

Set Parameters

void setParameters(const char* parameters)

Set parameters for the Agora Audio engine. The input argument is in JSON format specifying new parameters to set. Instead of being called directly by app, it is usually called by the helper class AgoraAudioParameters.

Name	Description
parameters	Parameters in JSON format: mute mutePeers speakerOn speakerVolume micVolume enableVolumeReport volumeSmoothFactor logFilter

Get Parameters

int getParameters(const char* parameters, char* buffer, size_t* length)

Retrieve current parameters settings.

Name	Description
parameters	Indicate which parameters to retrieve.
buffer	String containing values
length	Length of buffer

AgoraAudioParameters Methods

Mute

void mute(bool mute)

Turns off microphone.

Name	Description
mute	True turns off microphone. False turns back microphone.

Mute All Speakers

void mutePeers(bool mute);

Turns off both the speaker and earpiece.

Name	Description
mute	True turns off all audio output devices. False turns back on all audio output devices.

Mute Specific User

void mutePeer(bool mute, unsigned int uid);

Turn off audio for a specific caller.

Name	Description
mute	True means mute. False means unmute (i.e., turn back on caller's microphone.)
Uid	User to mute.

Select Speaker

void enableSpeaker(bool enable);

Name	Description
enable	False means output audio to earphone. True means output audio to speaker.

Set Speaker Volume

void setSpeakerVolume(int volume);

Name	Description
volume	Set volume from 0 (min) to max (255).

Set Microphone Volume

void setMicrophoneVolume(int volume);

Name	Description

volume	Set volume from 0 (min) to max (255).

IAgoraAudioEventHander Interface Methods

The callback methods in IAudioEventHandler are called when the user joins a call ro report on errors, success, and call quality. It calls these methods every 3 seconds for every user on the call.

onLoadAudioEngineSuccess

virtual void onLoadAudioEngineSuccess() = 0;

User implements this method to indicate what to do when the Agora audio engine is loaded correctly. This means the app was able to connect to an available audio server. From this point the audio engine is working, meaning it is in communication mode. Usually the app can start a time here to record the call duration.

onGetAudioSvrAddrSuccess

virtual void onGetAudioSvrAddrSuccess() = 0;

Notification callback.

onJoinSuccess

virtual void onJoinSuccess(unsigned int sid, unsigned int uid) = 0;

Indicates the client has logged into the server and the channel id and user id are allocated. The channel id is assign based on channel name specified by join() API. If the user id was not specified with the call to join(), the server will allocate one.

onError

virtual void onError(int rescode, const char* msg) = 0;

Name	Description
rescode	EVENT_LOAD_AUDIO_ENGINE_ERROR = 1001: failed to initialize audio engine. EVENT_START_CALL_ERROR = 1003: failed to start audio engine. Typically this is caused because the audio device is in use by another app. EVENT_JOIN_GET_AUDIO_ADDR_TIMEOUT = 11002: voice server list timeout. EVENT_JOIN_GET_AUDIO_ADDR_FAILED = 11003: error code received when requesting voice server list.

	EVENT_JOIN_GET_AUDIO_ADDR_ZERO_ADDR = 11004: The Voice Center Server (acts as a gateway, like DNS) responded that there is no voice server available. EVENT_JOIN_CONNECT_MEDIA_TIMEOUT = 12002: connect to voice server timeout. EVENT_JOIN_LOGIN_MEDIA_TIMEOUT_ALL = 13003: login voice server timeout. EVENT_JOIN_LOGIN_MEDIA_FAILED = 13004: Failed to login voice sever, server ACKed with error code EVENT_JOIN_LOGIN_REGET_AUDIO_ADDR = 13005: the voice central server (acts as a gateway, like DNS) tried all available voice servers but none is able to handle this call.
msg	Message that you want to pass to the method so that you can send it to the user interface or other.

onLogEvent

virtual void onLogEvent(const char* msg) = 0;

Log messages can be redirected to app instead of written to file. When enabled this function will be called to report log events. In other words this becomes the log handler.

Name	Description
msg	Log messages

onQuality

virtual void onQuality(unsigned int uid, unsigned short delay, unsigned short jitter, unsigned short lost, unsigned short lost2) = 0;

Name	Description
uid	User id, i.e. the caller.
rtt	Voice delay in ms.
jitter	Jitter means variation in the delay of received packets due to network congestion or queuing issues.
lost	Packet loss ratio.
lost2	Number of times that 2 consecutive packets were lost

onSpeakersReport

virtual void onSpeakersReport(const SpeakerInfo* speakers, unsigned int speakerNumber, int mixVolume) = 0;

Name	Description
speakers	An array containing current active speaker uid and volume (0-255) pairs.
speakerNumber	Length of speakers array.
mixVolume	Total volume, 0 to 255.

onLeaveChannel

virtual void onLeaveChannel(const SessionStat& stat) = 0;

Name	Description
stat	struct SessionStat { unsigned int duration; unsigned int txBytes; unsigned int rxBytes; };

onUpdateSessionStats

virtual void onUpdateSessionStats(const SessionStat& stat) = 0;

Name	Description
stat	struct SessionStat { unsigned int duration; unsigned int txBytes; // transmission unsigned int rxBytes; // receipt };

onAudioEngineEvent

virtual void onAudioEngineEvent(int evt) = 0;

Name	Description
evt	enum AUDIO_ENGINE_EVENT_CODE { AUDIO_ENGINE_RECORDING_ERROR = 0, // recording cannot proceed AUDIO_ENGINE_PLAYOUT_ERROR = 1, // player cannot proceed

	AUDIO_ENGINE_RECORDING_WARNING = 2, // other recorder related events
	
	AUDIO_ENGINE_PLAYOUT_WARNING = 3 // other player related events
	};

onAudioDeviceStateChanged

virtual void onAudioDeviceStateChanged(const char* deviceId, int deviceType, int deviceState) = 0;

Name	Description
deviceId	device id identifying an audio device

deviceType	enum AUDIO_DEVICE_TYPE {
	UNKNOWN_AUDIO_DEVICE = -1,
	PLAYOUT_DEVICE = 0,
	RECORDING_DEVICE = 1
	};

deviceState	enum AUDIO_DEVICE_STATE_TYPE {
	AUDIO_DEVICE_STATE_ACTIVE = 1,
	
	AUDIO_DEVICE_STATE_DISABLED = 2,
	
	AUDIO_DEVICE_STATE_NOT_PRESENT = 4,
	
	AUDIO_DEVICE_STATE_UNPLUGGED = 8
	};