

Omni Channel Management (OCM)

Tetherfi Livechat Mobile API Reference

Library version : 5.9.0

Last Updated Date : 2019 Dec 08

Version	Edited By	Date	Change
3.2	Nuwan	06/12/2018	Initial draft
3.4	Nuwan	17/12/2018	Config sections
4.0	Nuwan	11/02/2019	Video API Added Sequence Diagram added A reference to Livechat connect API added. React-native API Added
5.2	Nuwan	25/05/2019	Re-connect Handling description added,
5.3	Nuwan	31/05/2019	Self-view mirroring config added
5.4	Nuwan	12/06/2019	Background video handling functions added
5.9	Nuwan	08/12/2019	Document restructured and set in-built signaling as the default and recommended implementation

Contents

Introduction	3
Syntax, acromyms and abbrivations	3
Related components.....	3
Signalling.....	3
Signalling channel requirements	4
Signalling Methods	4
Integration artefacts.....	5
Supported versions	5
Permissions and modes required	5
Demo setup	5
Supported languages	6
Project Settings.....	6
Server integration API.....	6
Integration API (Inbuit signalling).....	7
Class : Livechat.....	7
Interface : SessionEventsHandler	10
Interface : AvCallEventsHandler	13
Configurations	15
Class : Configs	15
Livechat sdk media disconnect handling	17
(1) Handling events (Interface : AvCallEventsHandler).....	17
(2) Calling reconnect methods.....	18
Impact to application binary size.....	20
Appendix A : React-native API	20
Java script API	20
Video component	21
Appendix B : Native Api for external signalling.....	21
Class : RTCSession	21
Class / File Log.....	23
Class : VideoSessionFactory.....	23
VideoEventsListener	24
Class : AppIntegration.....	26
Signalling Messages	29

Introduction

This document is prepared to help developers to integrate with Tetherfi Live chat Library to implement chat channel with Mobile devices.

Syntax, acromyms and abbrivations



- Android specific code



- ios specific code

TMAC – Tetherfi Multimedia Agent Client

AV – Audio-Video

Signaling – Text message based communication method to control an AV session.

CSO – Customer service officer of the contact center

Related components

Below are the minimum required components for mobile chat channel implementation:

- 1) Text chat windows service (service to handle chat sessions)
- 2) Http proxy (for communications via internet)
- 3) Livechatconnect library (Connector for a third-party internal servers)
- 4) Mobile Libraries
- 5) TMAC server
- 6) TMAC client

In addition to above, below components are used in a production environment:

- 1) Audio-Video proxy
- 2) Media server
- 3) Recorder

Signalling

Audio – Video (referred hereafer as 'av') session control of the is done via text messages. This concept is referred here as signalling and channel being used for this communication is referred here as signalling channel.

Signalling channel requirements

Signalling channel should always fullfill below properties

- Reliability & sequence
- Full-duplex messaging
- HA support (Optional)

Signalling Methods

This Sdk supports 2 modes.

External Signaling – In this mode tetherfi-livechatav mobile library is used for av-capture and playback but signalling channel via internet is maintained by mobile app owners. Below are the properties of method.

- Mobile app already communicate with it's own server via internet which can be used for signalling covering above mentioned requirements.
- Mobile app is ready and expeting to use a 3rd party sdks like google firebase (<https://firebase.google.com/docs/cloud-messaging/>)
- Allow client organization to fully monitor information tetherfi exchange between mobile app and TMAC system.
- Internet port opening is only required for AV media channel (binary media data channel, most of the time UDP).
- Messaging path is : tehterfi-mobile-library > mobile app > mobile app server > tetherfi-livechatconnect > tetherfi chat server > tmac server > tmac client.

[Appendix A](#) describe further about this mode.

Inbuilt Signaling – In this mode tetherfi-livechatav mobile sdk with a suitable signalling implementation selected by tetherfi (depends on application architecture) is used for signalling. This can be used for mobile apps which has below properties.

- Standalone mobile app which does not have a corresponding server.
- Under exisisting implementation, server is not able to push a message mobile app. (e.g. app-server communication is request response based and server doesn't maintain a session)
- Tetherfi sdk will maintain siganlling via internet hence client organization has limited visibility and control over information exchanged in-between mobile app and tmac system.
- Internet port opening is required for 2 channels,
 - o For AV media channel (binary media data channel, most of the time UDP).
 - o TCP based signalling channel (could be http or websocket).

- Messaging path is : tetherfi-mobile-library <> tetherfi-chatproxy <> tetherfi chat server <> tmac server <> tmac client.

This is the default mode since version 6.0 which is described in below [sections](#).

Integration artefacts

- livechatav.aar, AppIntegration.java (Android library)
- livechatav.framework, AppIntegration.m/h (ios library)
- react-native-livechatav.tar.gz (react-native module for react applications)
- livechatconnect.jar (for external signalling in Signalling)

Supported versions

- Android 7 (M) +
- ios 11 +

Permissions and modes required

ios :

Privacy – Camera usage
Privacy – Microphone usage
Background mode : ON

android :

```
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Demo setup

ios: <https://testflight.apple.com/join/XovqQOoe>

android : <https://play.google.com/store/apps/details?id=com.livechatapp>

TMAC: https://sit.tetherfi.com:16443/TMAC_UI/Login.aspx
LanID: siraj / saman / nuwan
Station : -any-
Password : -any-

Supported languages

Android : Java
ios : Objective-C
react-native : javascript

Kotlin and Swift implementations requires wrapping respective legacy implementations

Project Settings

ios:

Bitcode support = NO
Objective C++ support

Server integration API

Java implementation of the API is available as livechatconnect.jar. which uses tcp/tls connections to connect with chat-service component of the TMAC.

Releated Java doc can be opened with below link

<http://demo.tetherfi.com:8081/livechatconnect/doc/>

Integration API (Inbuilt signalling)

[Class : Livechat](#)

```

🍏 public func SetSessionEventsHandler( handler:SessionEventsHandler )
🤖 public void SetSessionEventsHandler( Livechat.SessionEventsHandler
    handler )

```

Set chat session event listener. Refer [SessionEventsHandler](#).

```

🍏 public func Start( sessionDetails:SessionDetails, vaSession:String
    )
🤖 public void Start( SessionDetails sessionDetails, String vaSession )

```

Start an audio / video call, with Session Details. Response of this call is [OnStartSuccess](#) event for success or [OnStartFailed](#)

```

🍏 public func End( reason:String )
🤖 public void End( String reason )

```

End chat session (Including audio-video session if started). This call supersedes [EndAv](#).

```

🍏 public func SendMessage( msg:String )
🤖 public void SendMessage( String msg )

```

Send a text message for CSO agent to be displayed on agent's panel.


```

🍏 public func SendAppMessage( msg:String )
🤖 public void SendAppMessage( String msg )

```

Sends a custom application level message to TMAC. Format must be JSON with a type attribute.
 { "type": "xxx", }

```

🍏 public func NotifyTypingStateChange( state:Int )
🤖 public void NotifyTypingStateChange( int state )

```

Typing state indication , 0 – User is typing, 1 – Not typing

```

🍏 public func SetAvEventsHandler( handler:Livechat_AvCallEventsHandler
)
🤖 public void SetAvEventsHandler( Livechat.AvCallEventsHandler handler
)

```

Event Listener for audio-video events , refer [AvCallEventsHandler](#) .

```

🍏 func StartAudioCall()->Bool; func StartVideoCall()->Bool
🤖 boolean StartAudioCall(); boolean StartVideoCall();

```

Start an audio or video call locally

```

🍏 public func EndAvCall( reason:String )
🤖 public void EndAvCall( String reason )

```

End audio – video call

```

🍏 public func RequestAvCall( avmode:String )
🤖 public void RequestAvCall( String avmode )

```

Request to start an audio – video call by sending a message internally

avmode : (“audio” | “video” | “false”)

```

🍏 public func TryReconnect()
🤖 public void TryReconnect()

```

Application can request to renegotiate after a network change. However in order this to be effective signalling channel has to be online, which can be detected with [OnSignallingStateChanged](#) event.

```

🍏 public func PauseAvCall(); public func ResumeAvCall();
🤖 public void PauseAvCall(); public void ResumeAvCall();

```

Pause / Resume avCall with music

```

🍏 func AvResponse( avmode:String )
🤖 void AvResponse( String avmode )

```

Application call this function in order to inform response for [OnRequestAvCall](#) event of AvCallEventsHandler

```

🍏 func MuteAudio();public func UnmuteAudio();
🤖 void MuteAudio(); public void UnmuteAudio();

```

Mute/Unmute an audio / video call,

Interface : SessionEventsHandler

```

🍏 Livechat_SessionEventsHandler
🤖 Livechat.SessionEventsHandler

```

```

🍏 func OnStartSuccess(sid:String, tchatid:String, param:NSDictionary
);
🤖 void OnStartSuccess(String sid, String tchatid, String param)

```

Indicates chat session start in text mode. This is the success response for StartAvCall api

```

🍏 func OnStartFailed(errorCode:String);
🤖 void OnStartFailed(String errorCode);

```

Faliure response for StartAvCall api

```

🍏 func OnMessageReceived(agent:String, msg:String );
🤖 void OnMessageReceived( String agent, String msg );

```

On new text message from agent.

```

🍏 func OnEnd( initiator:String, reason:String );
🤖 void OnEnd(String initiator, String reason);

```

Complete end of interaction with CSO.

```

🍏 func OnRemoteUserConnected(name:String, agentId:String,
sessionId:String );
🤖 void OnRemoteUserConnected( String name, String agentId, String
sessionId);

```

Indicate connecting an agent. This event can be invoked multiple times as new agents connect.

```

🍏 func OnCallQueued( position:String, extra:String? );
🤖 void OnCallQueued( String position, String extra );

```

Indicate successful enqueuing of a session and waiting for an available CSO agent to connect.

```

🍏 func OnStatusNotification( notification:String );
🤖 void OnStatusNotification( String notification );

```

Status messages about the session in progress. Most of these messages indicate not being able to process due to a certain limitation /error. Default action of the application should be to end the session showing a suitable message to customer (depending on notification code).

```

🍏 func OnSignallingStateChanged( state:String );
🤖 void OnSignallingStateChanged( String state );

```

Indicate a temporary network availability changes , “online | offline”

```

🍏 func OnConferenceUserLeft( agent:String, msg:String );
🤖 void OnConferenceUserLeft( String agent, String msg );

```

Indicate a CSO agent being exit from ongoing conference

```

🍏 func OnTypingStateChanged( agent:String, state:Int );
🤖 void OnTypingStateChanged( String agent, int state );

```

Indicates typing state of th CSO (agent) at other side

```

🍏 func OnCallbackRequestStatus( isCustomReq:Bool, status:Bool);
🤖 void OnCallbackRequestStatus( boolean isCustomReq, boolean status );

```

Response for Callback request

Interface : AvCallEventsHandler

🍏 Livechat_AvCallEventsHandler

🤖 Livechat.AvCallEventsHandler

🍏 **func** OnRequestAvCall(avmode:String);

🤖 **void** OnRequestAvCall(String avmode);

Indicate an CSO side request for an avcall. Depending on the value, application is expected to popup a message to user asking to initiate an audio-video session.

🍏 **func** OnRespondAvCall(avmode:String);

🤖 **void** OnRespondAvCall(String avmode);

Indicate that CSO agent has responded to audio video request, initiated with StartAvCall.

🍏 **func** OnLocalVideo(localVideoView:UIView);

🤖 **void** OnLocalVideo(SurfaceView localVideoView);

An event from SDK, with self view from front camera.

localVideoView – Self view UI component. Application is expected to layout and add this to application's GUI.

```

🍏 func OnRemoteVideo( remoteVideoView:UIView );
🤖 void OnRemoteVideo( SurfaceView remoteVideoView );

```

An event from SDK with UI component of CSO's view

remoteVideoView - CSO's view as a UI component. Application is expected to layout add this to applications GUI.

```

🍏 func OnAvCallStartSuccess();
🤖 void OnAvCallStartSuccess();

```

Indicate establishing media connectivity with audio & video. After this both parties are able to see each other.

```

🍏 func OnAvCallStartFailure( if_any:Error? );
🤖 void OnAvCallStartFailure( Exception if_any );

```

Indicates not being able to establish audio-video connectivity (Post calling StartAudioCall or StartVideoCall)

```

🍏 func OnMediaDisconnect();
🤖 void OnMediaDisconnect();

```

Indicate a temporary media disconnect

🍏 **func** OnMediaReconnect();

🤖 **void** OnMediaReconnect();

Indiate a temporary media reconnect

🍏 **func** OnMediaReconnectableEnd();

🤖 **void** OnMediaReconnectableEnd();

Indiate a media disconnect, which is not conclusive. Application can invoke TryReconnect and wait for either OnMediaReconnect or OnEnd events.

🍏 **func** OnAvCallEnd(type:EndType, reasonDesc:String);

🤖 **void** OnAvCallEnd(RTCSession.RTCEventsHandler.EndType type , String reasonDesc);

Indicates the end of audio-video call. However the chat session is still active with CSO in text mode.

Configurations

[Class : Configs](#)

Singleton class to hold configurations.


```

🍏 -(void) addAvProxy:(NSString*)url
    user:(NSString*)username pass:(NSString*)passwd;

🤖 public void addAvProxy( String addr, String username,
    String password )

```

Add proxy url and credentials

```

🍏 @property enum AVMode avmode

🤖 public AVMode avmode

```

Set the mode to AUDIO or VIDEO. If the mode is video, VideoSessionFactory must have been initialized with a VideoEventListener. AppIntegration (if used) performs this step at setVideoEnabled method.

```

🍏 @property struct VideoProperties videoProps

🤖 public VideoProperties videoProps;

```

Capture height , width (in pixels) and frame rate (in fps) is specified here.

```

🍏 @property NSString* pauseAudioSoundFile;

🤖 public void setPauseAudio( int resourceId )

```

Optional configuration to play an mp3 audio file,

Add mp3 audio file to the main bundle of the application and set path in configurations by calling below method.

```

[[Configs Instance]
setPauseAudioSoundFile:
@"and_i.mp3"];
// file : and_i.mp3 is at the top
level of main application bundle

```

Create a raw resource folder if does not exist already.

Add mp3 audio file to resources of the application and set resource Id for in the configurations by calling below method.

```

Configs.Instance.setPauseAudio(
R.raw.andi );
//andi is the resouce id of

```

```

@property BOOL mirrorSelfview

public boolean mirrorSelfview;

```

Capture height , width (in pixels) and frame rate (in fps) is specified here.

Livechat sdk media disconnect handling

* This sections is to be updated for inbuilt signalling api.

Media disconnect handling involves below two things application is expected to perform.

1. Handling events
2. Calling reconnect method

(1) Handling events (Interface : AvCallEventsHandler)



```

interface Livechat.AvCallEventsHandler
{
    ---
    ---
    void OnAvCallStartSuccess();
    void OnMediaDisconnect();
    void OnMediaReconnect();
    void OnMediaReconnectableEnd();
    void OnAvCallEnd( EndType type,
String
    reasonDesc );
}

```



```

protocol Livechat_AvCallEventsHandler:
{
    ---
    ---
    func OnAvCallStartSuccess();
    func OnMediaDisconnect();
    func OnMediaReconnect();
    func OnMeidaReconnectableEnd();
    func OnAvCallEnd( type:EndType,
reasonDesc:String );
}

```

Above interface contains all events available for application. An object implementing this interface is passed to **RTCSession** class's **start** method which is a mandatory parameter.

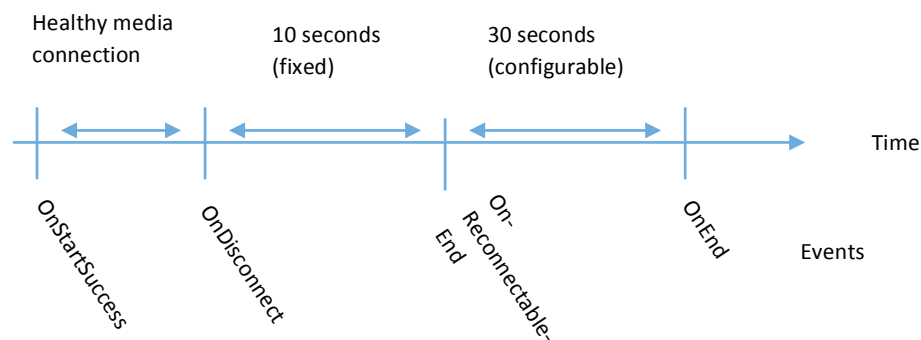
(2) Calling reconnect methods

```
Livechat.TryReconnect( );
```

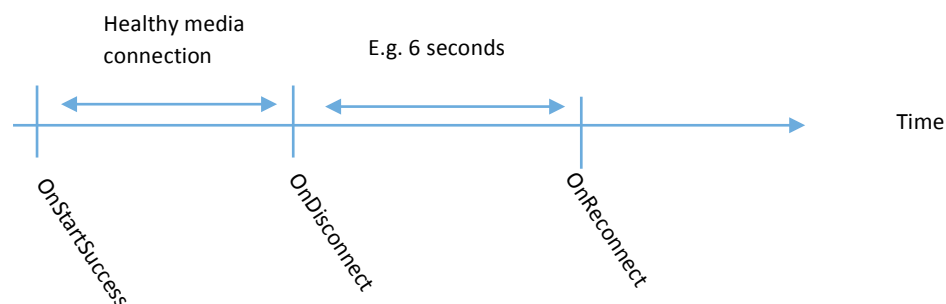
Application can call any of the above functions to initiate a reconnect attempt within correct time windows defined and explained below.

Below set of timeline illustrations shows order of events in each possible in each scenario.

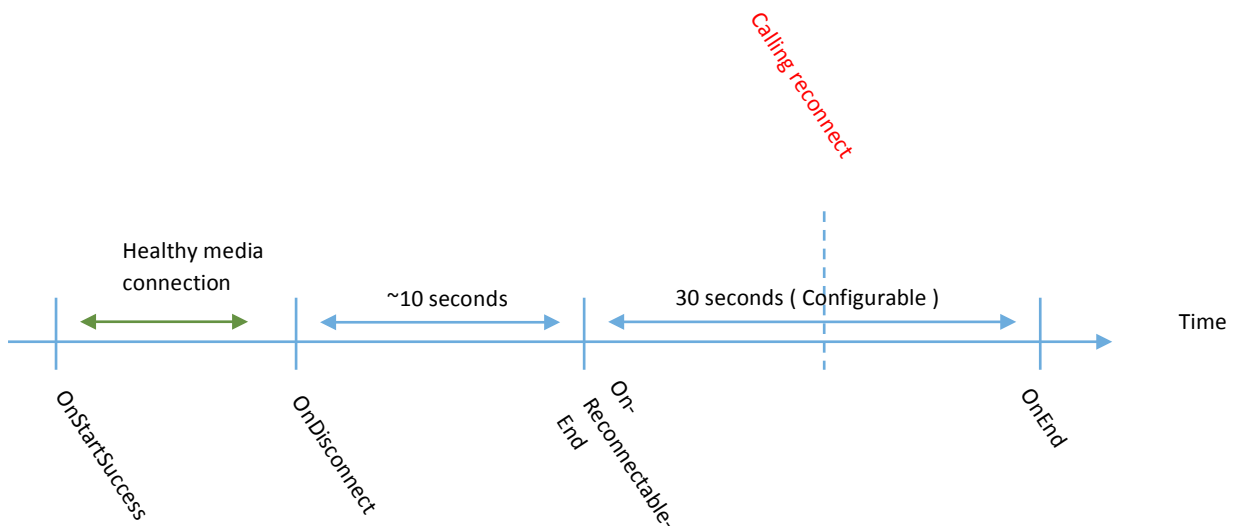
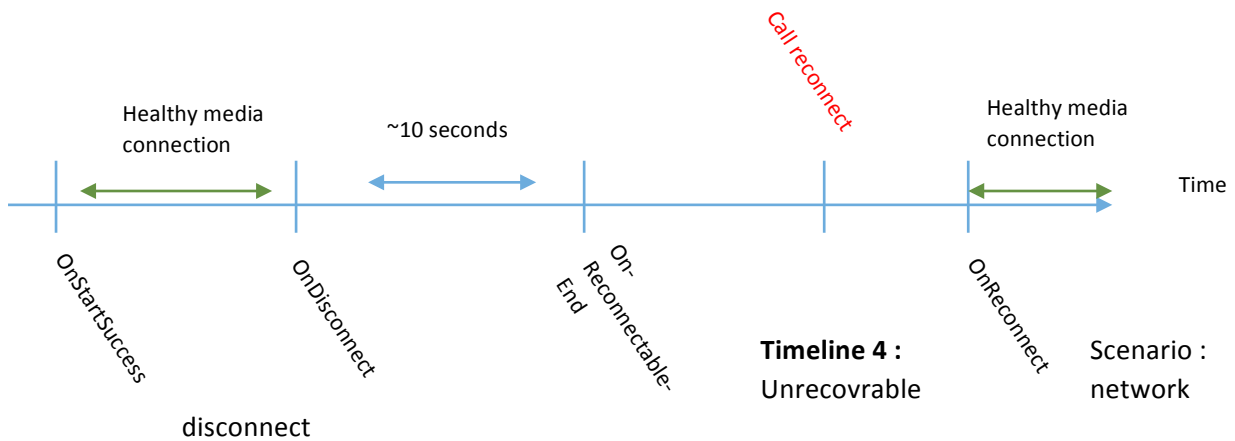
Timeline 1 : Basic flow



Timeline 2 : Temporary disconnect less than 10 seconds (No reconnect is expected).



Timeline 3 : Scenario : Disconnect lasts between approx. 10 - 40 seconds and application invokes reconnect within the window.



Notes:

- Between **OnDisconnect** and **OnReconnect** UI can show a suitable notification
- Calling reconnect method prior to **OnReconnectableEnd** event is possible and usually helpful in handling network switch.
- In order to recover disconnects due to network switching, calling '**reconnect**' is required. This type of disconnects will not recover automatically.
- Should not call reconnect method after **OnEnd**. Only option available at this point is calling **RTCSession.start**
- Reconnect function rely on signalling channel, if application is aware that signalling channel is also offline, it should not call for reconnect.
- Automatic reconnect of less than 10 seconds may not work in tcp mode. Recommended protocol is udp.
- **OnEnd** event is not limited to network issues (E.g. External gsm calls also can trigger **OnEnd**)

Impact to application binary size

This SDK consist with native libraries. which will increase the size of application binary size in following approximate amounts. Please note that, these number are mere approximates exact number variies between releases.

	Android	ios
Appstore Upload size	+ 35 MB	+ 20 MB
*Donwload size	+ 15 MB or +35MB	+ 20 MB
*Installed size	+ 15 MB	+ 20 MB
+File size for dev project	+35 MB	+35 MB

* Only these values impact mobile application user (customer)

+ This size of the total binaires distributed in this sdk

Appendix A : React-native API

Java script API

```
import { NativeModules, NativeEventEmitter } from 'react-native';
const { RNLivechatav } = NativeModules;
```

```
export default class AVChatConnection
{
  onLogMessage = ( type, class1, msg, time )=>{};
  startav( sid, onComplete ){}
  passSigMessage( message ){}
  endav( reason ){}
  setConfig( configs ){}
  micMute( doMute ){}
  speakerPhone( enable ){}
  pause(){}
  resume(){}
  playAudio(){}
  stopAudio(){}
  enableLogEvents( val ){}
  setAppName(name){}
  getAppName(){}
  reconnect(){}

  onSignallingMessage = (message)=>{};
  onEnd = ( type, desc ) =>{}
  onVideo = ( videoview ) =>{}
}
```

Video component

```
const Lvchatvideo = requireNativeComponent('RNVideoView', null);
<Lvchatvideo viewindex={this.remoteViewIndex} style={{ flex: 1, alignSelf: 'stretch' }} />
```

Appendix B : Native Api for external signalling

Class : RTCSession

Represent a one call session. App can use AppIntegration class (In External Signalling mode) or RTCSignallingSession (In Builtin signalling mode).

```
🍏 -(id _Nonnull) init;
🤖 public RTCSession( android.content.Context appContext )
```



Creates RTCSession object. Android version requires passing context of the application.

```
🍏 -(BOOL) start:(id<Signalling> _Nonnull) signaling
    forSession:(NSString* _Nullable) sid
    completionHandler:(nullable CompletionHandlerType) completionHandler;
🤖 public boolean start( Signalling externalSignalling,
    String sessionId, RTCEventsHandler compHandler )
```



Start a new audio / video call (depends on Configs.avmode).

Provided Signalling object is used to out signalling messages. Implementation should take care about delivering these messages to chat-server.



RTCEventsHandler get and async event on whether call started or not. Additionally it has onEnd callback which will be invoked if livechatav decide to end call session.

 `-(void)end:(NSString*_Nullable)desc;`
 `public void end(String desc)`



Ends ongoing av session, reason is a human readable string stating cause of end

 `-(void)onSigMessage:(NSString*_Nonnull)msg;`
 `public void onSigMessage(String msg)`





Function to feed incoming signalling messages from tmac into livechatav

 `-(void)changeSpeaker:(BOOL)isLoud;`
 `public void changeSpeaker(boolean isLoud)`

Tottle sound output device between loud speaker and earpiece / headset.

 `-(void)muteMic:(BOOL)set;`
 `public void muteMic(boolean set)`

Mute customer's voice

 `-(void)pause:(BOOL)withMusic ;`
 `-(void)resume;`
 `public void pause(boolean withMusic);`
 `public void resume()`

Hold/Unhold call sessions. Mutes and starts playing a stored mp3 audio file

🍏 `-(void)pauseVideo; -(void)resumeVideo;`

🤖 `void pauseVideo(); void resumeVideo()`

Pause and Resume only video streams. No effect for audio. Application is expected to call these functions as it goes to background and come back to foreground.

[Class / File Log](#)

🍏 `void set_livechat_logger(void(*func)(NSString* format, ...), BOOL includeTime)`

🤖 `public static void SetLogger(Logger lgr)`

Set an external logger in addition to default Android and ios loggers

🍏 `logi, logw, loge(NSString* str)`

🍏 `logip, logwp, logep(NSString* format, . . .);`

🤖 `Log.i, Log.e, Log.w (String msg)`

These are the functions used by sdk for logging. Application can use them for integration related log messages.

[Class : VideoSessionFactory](#)

🍏 `-(BOOL) initialize:(VideoEventsListener*)listener;`


```

🤖 public boolean initialize (VideoEventsListener
vEventsListener, Activity activity)

```

Initialize by session event handlers. Android activity instance is not required ('null') in this version.

```

🍏 -(void) pauseVideo;
🍏 -(void) resumeVideo;

🤖 public void pauseVideo();
🤖 public void resumeVideo();

```

Pause and Resume only video streams. No effect for audio.

[VideoEventsListener](#)

This is an interface class. Its implementation is a normal class with set of function blocks references.

```

🍏 typedef void(^OnLocalViewEvent)( UIView* view );
🍏 @property OnLocalViewEvent onLocalVideo;

🤖 void onLocalVideo( SurfaceView view)

```

Event carrying to self camera view (front facing)

```

🍏 typedef void(^OnRemoteViewEvent)( ViewEvent* viewEvent );
🍏 @property OnRemoteViewEvent onRemoteVideoAdded;

🤖 void onRemoteVideoAdded( SurfaceView view);

```

Event carrying CCO's (agent's) video view.

```
iOS : UIView* view = [ viewEvent createView:CGRectMake(0, 500, 500, 300) ];  
( Above line should be execute from application's main thread )
```

```
🍏 typedef void(^OnLocalViewEvent)( UIView* view );  
🍏 @property OnLocalViewEvent onRemoteVideoRemoved;  
🤖 void onRemoteVideoRemoved( SurfaceView view);
```

Event correspond to remote view removed

Class : AppIntegration

This class is referred only in **External signalling mode** , and distributed in the source form allowing to change it's content. This class is a sample code on how to use other related public classes in the binary (**RTCSession, VideoSessionFactory and VideoEventsListener**). Also please refer comments in AppIntegration.java file (which are applicable to ios file AppIntegration.m as well).

```

🍏 +(BOOL)startav:( NSString* )sessionId {. . .}

🤖 public static boolean startav( String sessionId ){. . .}

```

Start an audio / video call,

```

🍏 +(void)endav:( NSString* ) desc {. . .}

🤖 public static void endav( String desc ){. . .}

```

Ends the call. This function invokes RTCSession.end and also pass 'endav' message to tmac indicating the end of session.

```

🍏 +(void)setUserResponseInvoker:(void(^)(NSString*
sessionId, NSString* reqParam))invoker;

🤖 public static void setUserResponseInvoker(
IUserResponseInvoker invoker );

```

Set this callback function / interface to create a popup box asking user, whether to start an avsession (capturing microphone and camera) .

invoker - Library will invoke this function everytime if tmac request to upgrade and exiting text chat session to audio/video mode.

```

🍏 +(void)setUserResponse:( NSString* )sessionId
    answer:(BOOL)isAgreed req:( NSString* )reqParam;

🤖 public static void setUserResponse( String sessionId,
    boolean answer, String reqParam );

```

This is where user's response is set related to the popup question mentioned in previous method.
 Answer : **true / YES** for proceed with call **false / NO** to cancel
 reqParam : parameter from previous method.

Signaling message handling functions

Next two methods handle **IN** and **OUT** of signalling messages.

```

🍏 +(void)processAppMessageFromServer:( NSString* )sessionId
    msg:(NSString*)message;

🤖 public static void processAppMessageFromServer( String
    sessionId, String message );

```

To receive signalling messages from server or app.

```

🍏 static void passAppMessageToServer( NSString* sessionId,
    NSString* message )

🤖 private static void passAppMessageToServer( String
    sessionId, String message )

```

To pass signalling messages to server

```

🍏 +(void)setSessionObject:(id)obj;

🤖 public static void setSessionObject( Object obj );

```

Optional function to set a temporary object

```
🍏 +(RTCSession*)getActiveRTCSession;  
🤖 public static RTCSession getActiveRTCSession();
```

Return active RTCSession if any, else null or nil

Video Integration

```
🍏 static void setVideoEnabled( VideoEventsListener*  
    listener )  
🤖 public static void setVideoEnabled( VideoEventsListener  
    listener )
```

Function to initialize video by setting necessary callbacks. This one invokes, 'Class : VideoSessionFactory.Instance.initialize' method.

Signalling Messages

