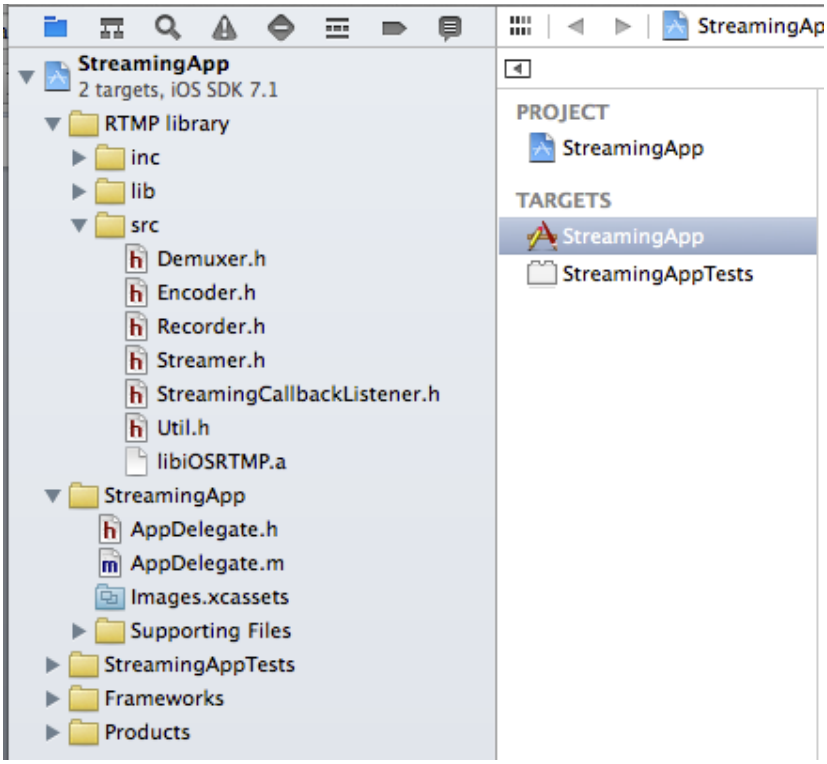
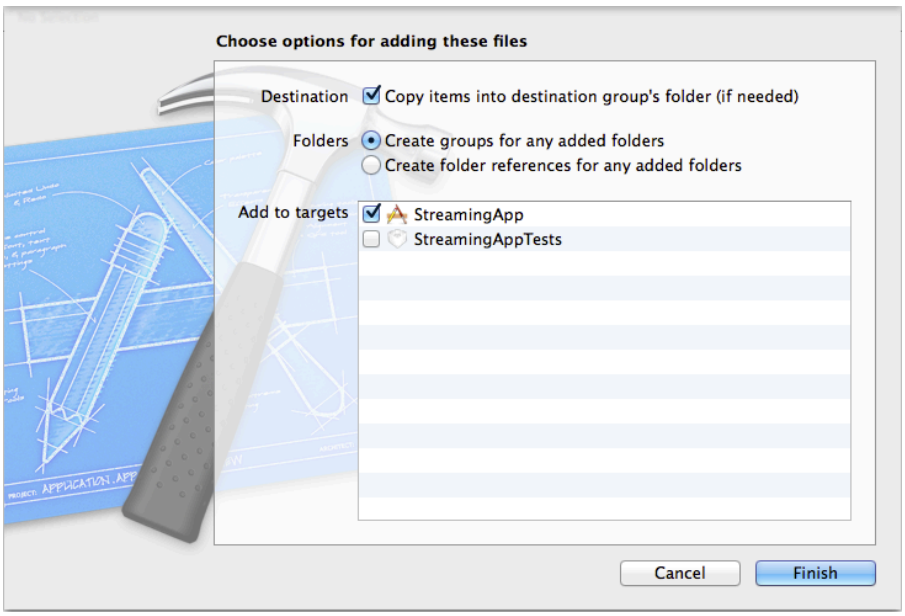
iOS RTMP library Setup

1. Add the library files

- copy the library folders (inc, lib, src) into your project's root folder (at the same level with the .xcodeproj file)
- drag the three folders inside your Xcode project

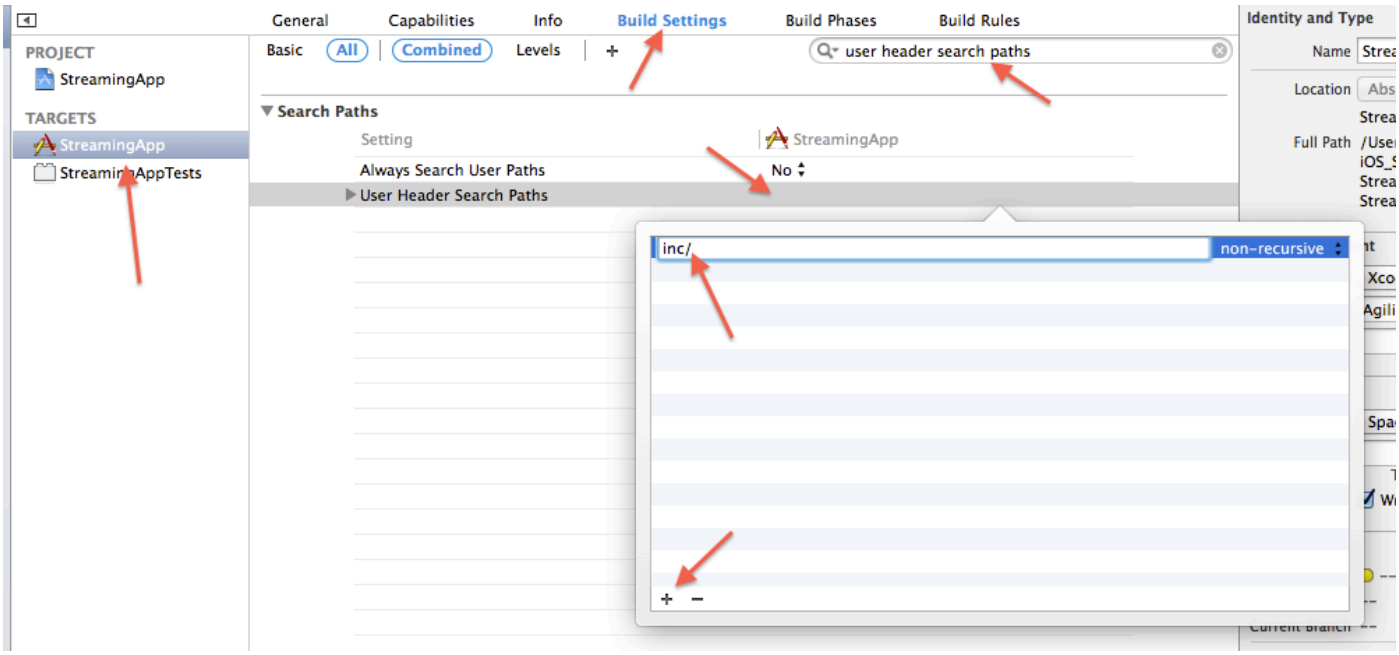


You can choose to drag the files directly inside the Xcode project without previously copying them in the root folder. Make sure you select the **"Copy Items into destination group's folder (if needed)"** checkbox.



2. Add User Header Search Paths

Go to the project's target then select **Build Settings.** Search for "User Header Search Paths" then add "inc/" to this section.
If the "inc" folder is not in the root folder, add the relative path as well (e.g. StreamingApp/inc/ instead of just inc/). This might be the case if you choose the second method at step 1.

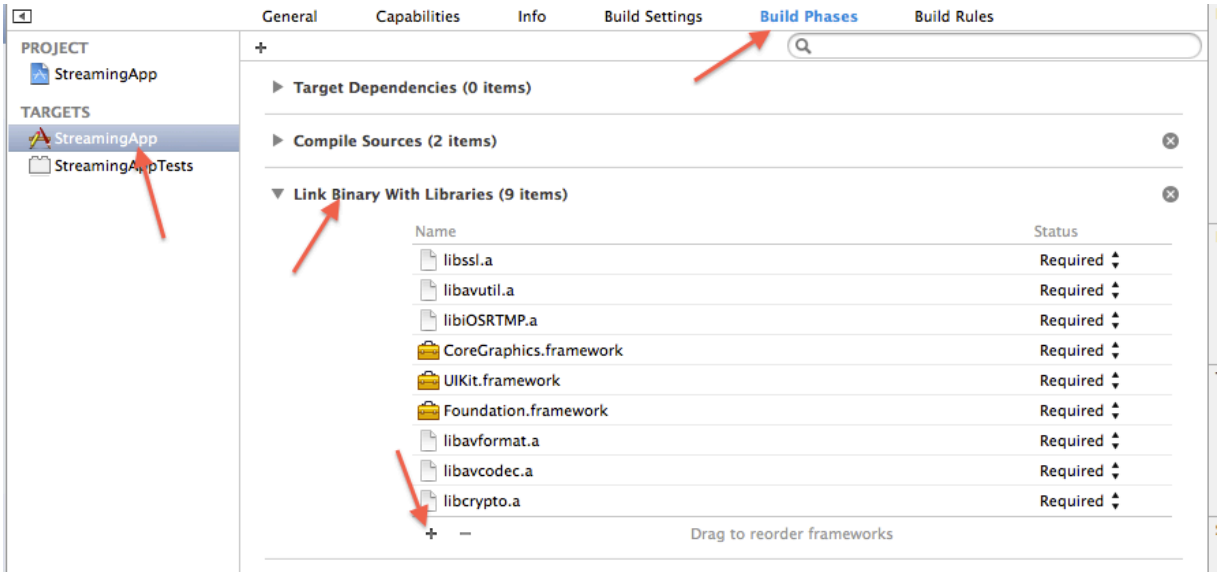


3. Additional frameworks

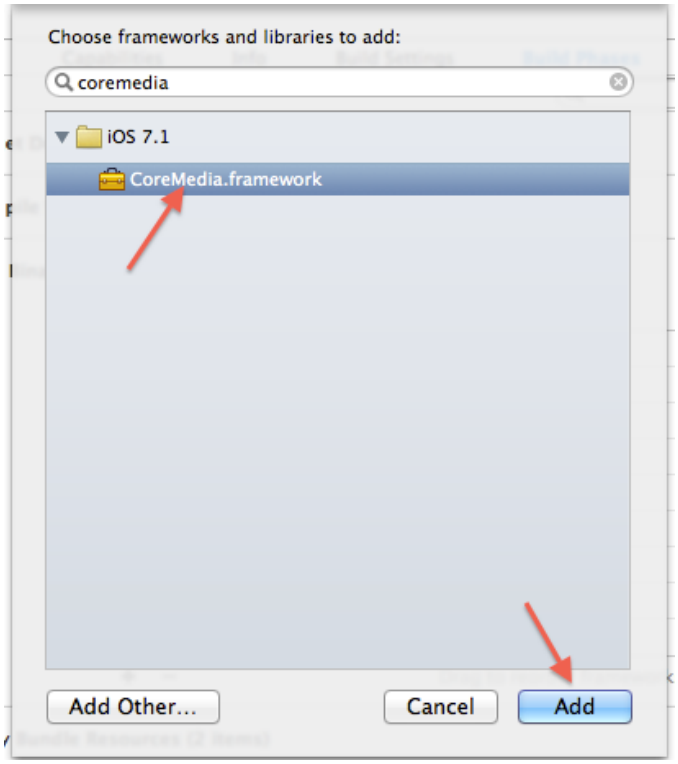
Add the following frameworks and libraries:

- CoreMedia.framework
- QuartzCore. framework
- AssetsLibrary. framework
- SystemConfiguration. framework
- AVFoundation. framework
- Libz.dylib library

Go to the project's target then select **Build Phases.** Expand **Link Binary With Libraries** and press the "+" button.



Add the above frameworks one by one: select the desired framework, then press ”Add”.



4. Initialize the library

Import the Recorder.h header into the class that you want to use as the video support.

```
#import "Recorder.h"
```

Add the class as delegate to the library’s callbacks and declare a global instance of the Recorder class.

```
@interface GoLiveViewController ()<StreamingCallbackListener>
{
    Recorder *_recorder;
}
```

Initialize the recorder.

```
    _recorder = [[Recorder alloc] initWithServer:
@"rtmp://your_server/application_name/stream_name"
                                        username: @"username_or_nil"
                                        password: @"password_or_nil"
                                         preview: self.view
                                          volume: 1.0
                                callbackListener: self
                                usingFrontCamera: NO
                                      usingTorch: NO
                                videoWithQuality: some_resolution
                                       audioRate: 44100
                                 andVideoBitRate: bps
                                keyFrameInterval: 24
```

```
                                framesPerSecond: 30
                                   andShowVideo: YES
                              saveVideoToCameraRoll: NO
                                 autoAdaptToNetwork: YES
                           allowVideoResolutionChanges: YES
                                   bufferLength: kDefaultBufferLength
                                validOrientations: Orientation_All
                              previewOrientation:
                                                Orientation_LandscapeRight];
```

5. Implement the protocol's methods

Implement the `StreamingCallbackListener`'s methods.

This methos is called with every new state of the streaming process.

```
- (void)streamingStateChanged:(StreamingState)streamingState
withMessage:(NSString *)message
{
    //your implementation here
}
```

The sreaming states are defined within:

```
typedef enum {
    StreamingState_Error = 0,
    StreamingState_Warning,
    StreamingState_Ready,
    StreamingState_Starting,
    StreamingState_Started,
    StreamingState_Stopping,
    StreamingState_Stopped
}StreamingState;
```

This method is called with every new modification in the video's properties: video frame rate, video bit rate, video resolution.

```
- (void)adaptedToNetworkWithState:(AutoAdaptState)adaptState
fps:(int)fps bitrate:(int)bitrate
resolution:(VideoResolution)resolution
{
    //your implementation here
}
```

The adapting states are defined within:

```
typedef enum {
    AutoAdaptState_DecreasingBitrate,
```

```
    AutoAdaptState_IncreasingBitrate,
    AutoAdaptState_DroppingVideoFrames,
    AutoAdaptState_IncreasingVideoFrames,
    AutoAdaptState_DecreasingResolution,
    AutoAdaptState_IncreasingResolution
}AutoAdaptState;
```

6. Start / Stop stream

Starting and stopping and stopping the stream is as simple as calling these two methods:

```
[_recorder start]; and [_recorder stop];
```

7. Releasing the Recorder

Make sure you only release the Recorder instance when it is ready to be released. The Recorder uses an AVCaptureSession instance and queues for processing the samples. Releasing the Recorder instance while still processing might throw exceptions or even cause crashes.

In conclusion, make sure you take these cases in consideration:

- if the session is running ([_recorder start] was called), only call release on the recorder instance after stopping the session ([recorder stop]) and make sure that `streamingStateChanged:withMessage:` is called with the `StreamingState_Stopped` state.
- After initializing the _recorder, make sure that `streamingStateChanged:withMessage:` is called with the `StreamingState_Ready` state before making any action on the _recoder (actions like start, release etc)

```
- (void)streamingStateChanged:(StreamingState)streamingState
withMessage:(NSString *)message
{
    switch (streamingState) {
        case StreamingState_Error:
            //return to initial state. An error occured and it's
no longer streaming.
            break;
        case StreamingState_Stopped:
            //it is now safe to release the _recorder (or restart)
            break;
        case StreamingState_Ready:
            //the _recorder is ready for use (or release)
            break;
        default:
            break;
    }
}
```

Using the library

1. Initialization methods

The library offers three different constructors for:

- live recording video
- video saved within the app's documents
- video from the Photo Library

Live recording video constructor

```
- (id)        initWithServer: (NSString *)server
                    username: (NSString *)username
                    password: (NSString *)password
                     preview: (UIView *)preview
                      volume: (CGFloat)initialVolume
             callbackListener:
(id<StreamingCallbackListener>)callbackListener
             usingFrontCamera: (bool)front
                   usingTorch: (bool)torch
             videoWithQuality: (VideoResolution)videoResolution
                    audioRate: (double)audioRate
              andVideoBitRate: (NSInteger)videoBitRate
             keyFrameInterval: (NSInteger)keyframeInterval
              framesPerSecond: (NSInteger)fps
                 andShowVideo: (bool)showVideo
       saveVideoToCameraRoll: (bool)saveVideo
          autoAdaptToNetwork: (bool)autoAdapt
allowVideoResolutionChanges: (bool)allowChanges
                bufferLength: (double)bufferLength
           validOrientations: (Video_Orientation)orientation
          previewOrientation: (Video_Orientation)previewOrientation;
```

@param *server*

This is the RTMP service full path: (**server** + **application** + **stream_name**). The path must begin with the protocol prefix: **rtmp://** or **rtmps://**. The library supports **secured** RTMP (through **SSL**) . Next is the server address, followed by the application and stream name. This is how it should look: **rtmp://server_address/application_name/stream_name.**
Here is an example of a valid parameter: **rtmp://fms4.modenacam.com/bogdan/myStream**. You can actually use this link for testing.

@param *username*

The RTMP authentication **username** if it is required. The library supports streaming to RTMP destinations that require authentication. If no username is required **Nil** should be passed.

@param *password*

The RTMP authentication **password** if it is required. The library supports streaming to RTMP destinations that require authentication. If no password is required **Nil** should be passed.

@param *preview*

A **UIView** instance where the video preview will be inserted as a subview. One can pass the **self.view** property of the **UIViewController** that is used as the video support.

@param *initializeVolume*

A float value representing the default sound volume. The value must be between 0.0 and 1.0. It can be modified after the initialization even while streaming.

@param *callbackListener*

The class implementing the streaming delegate methods.

@param *front*

BOOL value stating whether the default camera should be the front one or not. If YES, the default camera will be the front (FaceTime) camera. If NO, the back camera will be used.

@param *torch*

BOOL value stating whether the device's torch should be On or not. If YES, the torch will be On.

@param *videoResolution*

The video resolution. The value passed here must be within the **VideoResolution** enumeration which can be found in the **Util.h** file.

The **VideoResolution_1920x1080** is only available as an **extra feature**. If you don't have this extra feature and pass **VideoResolution_1920x1080, VideoResolution_192x144** will be used with the MaximumBitrate_192x144 regarding of what you've passed for the **videoBitRate** parameter.

@param *audioRate*

The audio rate (frequency) measured in Hz. We strongly recommend the use of kDefaultAudioRate value (which is 44100 Hz).

@param *videoBitRate*

The video bit rate measured in bits per second (**bps**). You should choose the video bitrate based on the video resolution. That's why we recommend setting a value between MinimumBitrates and

MaximumBitrates enumerations accordingly to the video resolution. These enumerations are declared in the **Util.h** file.
E.g. for VideoResolution_1280x720 you should choose a video bitrate value between MinimumBitrate_1280x720 and MaximumBitrate_1280x720.
Of course, these are just suggestions, you can always pass your own values based on the device and network performances.

@param *keyframeInterval*

The video key frame interval. We recommend using kDefaulKeyFrameInterval.

@param *fps*

The number of frames per second. The maximum frame rate depends on the device's camera so be careful when passing the value. The iPhone 4S' camera delivers a maximum of 30 frames per second, while the iPhone 5S' camera can deliver 60 frames per second.

@param *showVideo*

A BOOL value stating whether the video frames should be sent or not. You can choose to stream only the audio data. You can change this anytime by calling "`showVideo:`", even while streaming.

@param *saveVideo*

The service can save the streamed video in the Photo Library. Be sure that the phone has enough storage space before setting this value to YES. Also, by saving the video might interfere with the device's performances, so make some tests first.

@param *autoAdapt* (extra feature)

If TRUE the library manager will adapt the video settings to the network and device's performances (upload speed, routing, device capabilities etc). If the network doesn't supply enough performance for the chosen configuration the manager will decrease the video bitrate, drop video frames or even decrease the video resolution. Based on the **videoResolution** parameter, the network manager will decrease the video bitrate with 25% until it reaches the minimum value for the selected video resolution from the MinimumBitrates enumeration. The next step is to drop video frames. At first every third video frame will be dropped. After that, every second video frame is dropped. If the **allowChanges** parameter is YES, the manager will decrease the video resolution and continue decreasing the bitrate until it reaches 192x144 and MinimumBitrate_192x144.

When the network improves its performance, the video settings start to gain bitrate, frames per second and higher resolution until is reaches the initial ones (not more than the ones passed initially).

If this feature is not included in your version, passing TRUE to this parameter will have no effect.

@param allowChanges

This parameter is only took in consideration if the **autoAdapt** is **TRUE** and the library version includes the extra feature. If **TRUE** will allow the network manager to decrease the initial video resolution when needed to adapt to the current network and device performances.

@param bufferLength

The library uses a buffer from which it streams the media data. Buffer length represents **half duration** (in seconds) of video being processed. The minimum value of the buffer's length depends on the video configuration (resolution, bitrate). The total latency of the stream will be given by the sum of double the buffer's length and the network delay. In conclusion, a small buffer will give us a reduced latency.

Why choose a length bigger than the minimum value?

The length of the buffer is also tied to the video resolution and bitrate. The device must be able to process the amount of data buffered in "length" seconds. If the device doesn't have the hardware configuration to process all the data from the buffer you might get errors like: "`Couldn't open file`"
The default buffer length is 1.5 seconds. This buffer will be enough for all resolutions up to 1920x1080 for most of the iOS devices.

We strongly recommend passing **kDefaultBufferLength**.

@param orientation

Represents the allowed video orientations. Must be a value within the Video_Orientation enumeration found in the **Util.h** file.
if you pass **VideoOrientation_Portrait**, it means that the video will only be recorded in portrait mode regarding the device orientation. But if you pass **VideoOrientation_All**, the service will be able to record the video in any orientation, based on the device orientation. So if the device is **LandscapeRight** oriented in the moment of calling the **start** method, the video will be recorded in that orientation. Please keep in mind that the service will record the video in the orientation of the device in the moment of starting the streaming. If the user tilt's the device while streaming, it will remain the same orientation until it stops the streaming and start again in a different orientation.

@param previewOrientation

Refers to the UIViewController orientation. In other words, the UI presentation orientation or the orientation of the UIViewController that is used as the video preview support. You should never pass VideoOrientation_All here. If your ViewController's user interface is designed as portrait, pass VideoOrientation_Portrait and so on.

Video from the Photo Library constructor

```
-   (id)initWithServer: (NSString *)server
              username: (NSString *)username
              password: (NSString *)password
    andLibraryFilePath: (NSURL *)thePath
   andCallbackListener: (id<StreamingCallbackListener>)callbackListener
         andShowVideo: (bool)showVideo;
```

@param *server*

This is the RTMP service full path: (**server** + **application** + **stream_name**). The path must begin with the protocol prefix: **rtmp://** or **rtmps://**. The library supports **secured** RTMP (through **SSL**) . Next is the server address, followed by the application and stream name. This is how it should look: **rtmp://server_address/application_name/stream_name.**
Here is an example of a valid parameter: **rtmp://fms4.modenacam.com/bogdan/myStream**. You can actually use this link for testing.

@param *username*

The RTMP authentication **username** if it is required. The library supports streaming to RTMP destinations that require authentication. If no username is required **Nil** should be passed.

@param *password*

The RTMP authentication **password** if it is required. The library supports streaming to RTMP destinations that require authentication. If no password is required **Nil** should be passed.

@param *thePath*

The url where the video can be found. Exactly how the Photo Library file picker returns it.

@param *callbackListener*

The class implementing the streaming delegate methods.

@param *showVideo*

A BOOL value stating whether the video frames should be sent or not. You can choose to stream only the audio data. You can change this anytime by calling "`showVideo:`", even while streaming.

Video saved within the app's documents constructor

```
-   (id)initWithServer: (NSString *)server
              username: (NSString *)username
              password: (NSString *)password
           andFilePath: (NSString *)thePath
   andCallbackListener: (id<StreamingCallbackListener>)callbackListener
          andShowVideo: (bool)showVideo;
```

@param *server*

This is the RTMP service full path: (**server** + **application** + **stream_name**). The path must begin with the protocol prefix: **rtmp://** or **rtmps://**. The library supports **secured** RTMP (through **SSL**) . Next is the server address, followed by the application and stream name. This is how it should look: **rtmp://server_address/application_name/stream_name.**
Here is an example of a valid parameter: **rtmp://fms4.modenacam.com/bogdan/myStream**. You can actually use this link for testing.

@param *username*

The RTMP authentication **username** if it is required. The library supports streaming to RTMP destinations that require authentication. If no username is required **Nil** should be passed.

@param *password*

The RTMP authentication **password** if it is required. The library supports streaming to RTMP destinations that require authentication. If no password is required **Nil** should be passed.

@param *thePath*

The string path to the video file from the app's documents.

@param *callbackListener*

The class implementing the streaming delegate methods.

@param *showVideo*

A BOOL value stating whether the video frames should be sent or not. You can choose to stream only the audio data. You can change this anytime by calling "`showVideo:`", even while streaming.

2. Library public methods

start

Starts the streaming.

- (void)start

stop

Stops the streaming.

- (void)stop

active

Returns whether the service is streaming or not.

- (bool)active

setVolume

Sets the audio volume. The passed value must be a float between 0.0 and 1.0

- (void)setVolume:(float)value

getVolume

Returns the current audio volume.

- (float)getVolume

showVideo

Tells the service to send the video frames or to only stream audio data. If YES is passed, the video frames will be sent. If NO is passed, only audio data will be streamed.

- (void)showVideo:(BOOL)show

useFrontCamera

Sets the active capture camera. If YES is passed, the service will start capturing with the front (FaceTime) camera. If NO is passed, the service will capture video using the back camera. You must make sure that the intended camera supports the current video resolution, otherwise it won't have any effect.
If the new camera can be used, it returns YES.

- (BOOL)useFrontCamera:(BOOL)front

useTorch

One may opt to use the device's torch. By passing YES, the torch will be set On. Call this method with NO to set the torch Off.

- (void)useTorch(BOOL)torch

3. Callback listener (delegate)

In order to properly use the library you must implement the **StreamingCallbackListener** methods.

streamingStateChanged:withMessage:

This method is called with every new state of the library.

```
– (void)streamingStateChanged: (StreamingState)streamingState
                  withMessage: (NSString *)message;
```

@param *streamingState*

Describes the new state of the library. It can take values within the **StreamingState** enumeration declared in the **Util.h** file.

StreamingState_Error

This value is passed when an error occurred. It may be passed while initializing the **Recorder** or while streaming. If this value is passed while **streaming** (after the **start** method was called) the streaming will be stopped, so you should expect the method to be called with **StreamingState_Stopped.** You must wait for the **StreamingState_Stopped** state before releasing the **Recorder** instance. The session needs to properly close.

StreamingState_Warning

This value is passed to inform about certain events. The service will continue running after this state, but be careful about the info it provides.

StreamingState_Ready

This value is passed to inform that the **Recorder** instance initialization was done properly and the service is ready to stream. **DO NOT** release the **Recorder** instance before this state.

StreamingState_Starting

This value is passed after the **start** method is called to inform that the library is preparing to start streaming.

StreamingState_Started

This value is passed when the library started encoding and streaming.

StreamingState_Stopping

This value is passed after the **stop** method is called to inform that the library is preparing to stop the streaming.

StreamingState_Stopped

This value is passed when the streaming stopped. It is now safe to release the **Recorder** instance. **DO NOT** release the **recorder** before this state (if the **start** method was previously called).

@param *message*

String describing the state.

adaptedToNetworkWithState:fps:bitrate:resolution

```
@optional
- (void)adaptedToNetworkWithState: (AutoAdaptState)adaptState
                              fps: (int)fps
                          bitrate: (int)bitrate
                       resolution: (VideoResolution)resolution;
```

This method is called only if the library contains the **extra feature** which allows it to auto adapt to network performances. It will be called every time the manager makes changes on the number of frames per second, bitrate and/or video resolution.

@param *adaptState*

Gives the network manager state which is a value from the **AutoAdaptState** enumeration. The manager can:

- increase / decrease video bitrate: **AutoAdaptState_IncreasingBitrate** or **AutoAdaptState_DecreasingBitrate** respectively
- increase / decrease the number of video frames per second: **AutoAdaptState_IncreasingVideoFrames** or **AutoAdaptState_DroppingVideoFrames** respectively
- increase / decrease video resolution: **AutoAdaptState_IncreasingResolution** or **AutoAdaptState_DecreasingResolution** respectively

@param *fps*

The current number of frames per second.

@param *bitrate*

The current video bitrate.

@param *resolution*

The current video resolution.