Apple MeetingSDK v0.6 Release Notes. (May 8, 2023)

NOTE:
Version 0.6 of the Visionable Apple MeetingSDK is the first official release that exposes functionality available in Visionable's "V3" server architecture. This SDK continues to provide an interface to the Visionable "V2" architecture as well.

OVERARCHING CHANGES
Starting with the transitional v0.5.x releases and completing with the v0.6 release, the Visionable MeetingSDK has been re-architected to rely on a layer of cross-platform C++-based code to manage the parsing of XML objects coming from our audio/video engine, for establishing model objects representing Meetings and Participants, and for firing "delegate methods/callbacks" notifying your application of changes in state for the current meeting. Prior to v0.5, these functions were individually implemented per-platform supported in that platform's native language. Moving all of this functionality into a common, C++ codebase should result in consistent behavior when dealing with Visionable back-end servers.

CONNECTING TO V3 SERVERS
In Visionable's V3 architecture, a special token (referred to as an MJWT token) is required to join a meeting. There are two types of MJWT tokens: a *guest* MJWT token that doesn't correspond to Visionable user and an *authenticated* MJWT token that is obtained by passing a JWT token obtained from Visionable's authentication system (not covered here). To retrieve an MJWT token, use the new `initializeMeetingWithToken` API call *instead of* the original `initializeMeeting` API call (which is used only with V2 servers).

```
public func initializeMeetingWithToken(meetingUUID: String,
server: String, token: String?, completion: @escaping
(Bool,String) -> ())
```

This function still takes a `meetingUUID` and a `server` name but now also takes a `token` parameter that is either `nil` if you wish to obtain a guest MJWT or it contains a JWT token if you want to obtain an authenticated MJWT.

The completion routine for `initializeMeetingWithToken` now is called with a second parameter (String) that contains the MJWT token (guest or authenticated).

Once you obtain an MJWT, you now join the meeting with a call to joinMeetingWithToken:

```
public func joinMeetingWithToken(server: String, meetingUUID:
String, token: String, userUUID: String = "", name: String,
completion: @escaping (Bool) -> ())
```

This function takes the `server` name you are connecting to, the `meetingUUID` for the meeting, the MJWT in the `token` parameter, an option `userUUID` to be associated with the user (pass an empty string to have the SDK generate a userUUID), and the `name` of the user to be shown in the meeting.

Using these two calls will allow you to connect to a V3 meeting. Once connected, all other SDK functionality is the same as with V2 servers.

CONNECTING TO V2 SERVERS

The APIs for connecting to V2 servers have changed slightly. The `initializeMeeting` API call now looks like this:

```
public func initializeMeeting(meetingUUID: String, server:
String, completion: @escaping (Bool,String) -> ())
```

The completion routine now is called with a second argument that is the AES256 encryption key used for the meeting. Previous SDKs just cached this internally, however now you need to receive it from `initializeMeeting` and pass it to the `joinMeeting` call.

The `joinMeeting` call now requires you to pass all connection parameters. If the "userUUID" parameter is an empty string, the SDK will generate a guest-based identifier to associate with this participant:

```
public func joinMeeting(server: String, meetingUUID: String,
key: String, userUUID: String = "", name: String, completion:
@escaping (Bool) -> ())
```

API CHANGES (since the last v0.4 release)

To retrieve a `Participant` object that represents the local user, use this method:

```
public func getLocalParticipant() -> Participant?
```

The `videoInfo` field of the `Participant` object is now a `Dictionary`. For all key-value pairs in the Dictionary, the "key" is a streamId and the value is the corresponding `VideoInfo` object.

The `MeetingSDK` singleton now has an accessor function named `getParticipants`() that returns an array of `Participant` objects (this used to be a simple property in previous versions of the SDK)

The VideoInfo structure has the following changes:

- The `active` property is now a `Bool`
- The `local` property is now a `Bool`
- The `width` property is now an `Int`
- The `height` property is now an `Int`
- The `layout` property is now an `Int`
- There is still a property named `videoView`, but we do not populate it at this time and it will always be `nil`.

New APIs have been added for supporting audio input/output preview. You can start and stop audio input and output preview with the following APIs. When audio input preview is enabled, the `MeetingSDKDelegate` method `inputMeterChanged` calls will occur. When audio output preview is enabled, the `MeetingSDKDelegate` method `outputMeterChanged` calls will occur.

```
public func enableAudioInputPreview(device: String)->Bool
```

Start audio input preview events. Input will be taken from the device specified which should be one of the strings returned by the `getAudioInputDevices` API call. For iOS, the only available device is called “Default device” and it will reflect whatever the input source is for the device (speakerphone mic, headset, etc.)

```
public func disableAudioInputPreview(device: String)->Bool
```

Stop audio input preview events from the device specified which should be one of the strings returned by the `getAudioInputDevices` API call. For iOS, the only available device is called “Default device” and it will reflect whatever the input source is for the device (speakerphone, headset, etc.)

```
public func enableAudioOutputPreview(device: String, soundPath:
String)->Bool
```

Start audio output preview events. Output will be played on the device specified which should be one of the strings returned by the `getAudioOutputDevices` API call. For iOS, the only available device is called “Default device” and it will reflect whatever the output source is for the device (speakerphone, headset, etc.)

The second argument, `soundPath`, is an absolute path to a .wav file that should be played to test output. For both iOS and MacOS, there are framework calls that will allow you to retrieve

an absolute path name to .wav files stored in your Xcode project. It is up to you to retrieve the absolute path and pass it as the second argument to this API call.

```
public func disableAudioOutputPreview(device: String)->Bool
```

Stop audio output preview events from the device specified which should be one of the strings returned by the `getAudioOutputDevices` API call. For iOS, the only available device is called "Default device" and it will reflect whatever the output source is for the device (speakerphone, headset, etc.)

New APIs have been added for supporting video preview. The following MeetingSDK API calls have been added:

```
public func enableVideoPreview(camera: String, withMode: String,
lowLevel: Bool, completion: @escaping (Bool) -> ())
```

Starts a video preview session with the specified camera. The camera `camera` parameter should be one of the values returned by `getVideoDevices`. The `withMode` parameter is one of the standard supported resolutions (same ones used for the `enableVideoStream` API call). The `lowLevel` flag is set to `true` if you only want to receive `NSData` block updates with raw frame data as opposed to having the `MeetingSDK` maintain a `UIView` for you (same as with the `enableVideoStream` API call). When the video preview has started, you will be notified via a new `MeetingSDKDelegate` routine, documented below.

```
public func disableVideoPreview(camera: String)
```

Stop video preview events from the device specified which should be one of the strings returned by the `getVideoDevices` API call.

```
func previewVideoViewCreated(videoView: VideoView)
```

A new method in `MeetingSDKDelegate` that is called when a video preview is started. The `videoView` parameter will always be populated. If the preview was enabled with the `lowLevel` flag set to `true`, the `videoView` will only contain updates to raw frame data via the `imageData` member . Otherwise `videoView` will be a UIView-based object that the SDK will render the corresponding video in (and you only need add it to your user interface's view hierarchy).

APIs for interfacing with audio devices have been given a device parameter where appropriate:

```
public func setAudioInputVolume(_ volume: Int32, device:
String)->Bool
public func setAudioOutputVolume(_ volume: Int32, device:
String)->Bool
```

The above APIs are only defined properly for MacOS. Single argument versions still exist that will operate on whatever the SDK thinks is the “current device”, but the two argument versions let you specify a particular device by name.

```
public func disableVideoPreview(camera: String)
```

Pass the name of the camera device you wish to disable preview functionality for.

Video codec constants have changed:

```
public enum CameraMode: String {
    /// Small Video
    case small = "SMALL"
    /// Medium
    case medium = "MEDIUM"
    /// Large Video
    case large = "LARGE"
    /// HD1 video (720p)
    case hd1 = "HD1"
    /// HD2 video (1080p) -- 1920 wide x 1080 tall
    case hd2 = "HD2"
    /// HD3 Video
    case hd3 = "HD3"
    /// 4K video (4K) -- 3840 wide x 2160 tall
    case fourK = "4K HIGH"
```

You must now use the above constants when specifying video send resolutions.

```
public var isScreenShare: Bool
```

Exposed `isScreenShare` value in all `VideoInfo` objects to be used to determine if a given `VideoInfo` object represents a screen share.

```
public func setAudioStreamVolume(streamId: String, volume:
Int32)->Bool
```

Updated API to return a Boolean to be consistent with similar API calls. Boolean is whether or not the request to set the volume was made successfully.

Re-introduced the `participantVideoViewRetrieved` callback:

```
func participantVideoViewRetrieved(participant: Participant,
                                   videoView: VideoView, local: Bool)
```

Called when a video stream that had previously been enabled and disabled is enabled again. In this case, the SDK does not destroy the VideoView that was created when the stream was initially enabled; it maintains a reference to it so that it can be re-used should the stream be re-enabled.

```
public func joinMeetingWithToken(server: String, meetingUUID:
String, token: String, userUUID: String = "", name: String,
completion: @escaping (Bool) -> ())
```

This API call hasn't changed, but the `userUUID` parameter is now ignored and will be removed in a future version. Since, for V3 connections, you are either passing an authenticated MJWT for authenticated users OR a guest MJWT by passing nil for the `token` parameter, there is no need to pass a userUUID. This is because the user's UUID is in the MJWT (either their authenticated unique identifier or a guest one). As such there is no need to pass it separately. The SDK will always generate a prefix for the UUID stored in the MJWT so that the user has a unique identifier per device if they are logged into multiple devices.

```
public func setLogFile(_ filename: String)
```

Allows you to provide an absolute path name (or path name relative to the directory the executable is launched from) of a file to be used to log IGAudio, IGVideo, CoreMeeting and MeetingSDK log entries. In the event of a crash, this file should be updated with the latest log entries before the application terminates.

This API should work for both MacOS and iOS, but with iOS the application would need to determine an appropriate absolute path name and then the application code would need to access the file (there is no way to access this file from outside of the iOS Application's sandbox). You *could* enable your application as one that "shares files" and then find the appropriate Documents directory for your application and provide a full path to a file in that directory (then, in theory, a user could access the log file from the finder when the iOS device is connected to the computer).

For MacOS, you need to find a directory that the application is allowed to write to. The only such directory guaranteed is the Mac's /tmp directory. You may be able to authorize your application to be able to write to other directories in which case you can then specify a file in such a directory for logging.

IMPORTANT: In order to capture log entries from the underlying audio and video engines, you still need to call `MeetingSDK.shared.enableInlineAudioVideoLogging(true)`.

NOTE: This logging *will* function even if the MeetingSDK delegate is not set.

A new `MeetingSDKDelegate` method has been added:

```
func previewVideoUpdated(streamId: String)
```

It is intended to be called when an active preview video stream is changed (such as when previewing an iPad based camera and the device is rotated).

CHANGES/FIXES

Changes fixes here detail what has been fixed since the last v0.5.19 release:

Fixed a problem that could cause crashes when a remote user using a web client enters the meeting.

KNOWN ISSUES

The new `previewVideoUpdated` delegate method may *not* be called when a device rotates. This will be resolved in a future SDK release.

In support of the new threading model, all delegate methods are executed on a serial OperationQueue that is created by the SDK. Future versions will allow you to specify an OperationQueue that you create (or use the main queue)

The `VideoView.isScreenShare()` API call will likely return `true` for non screen-share video streams if they are being sent at 4K resolution.