Apple MeetingSDK v0.5.3 Release Notes. (October 10, 2022)

NOTE:
Version 0.5.x of the Visionable Apple MeetingSDK is a transitional release that begins to expose functionality available in Visionable's "V3" server architecture. As this architecture is not expected to be in production until 4Q2022, any 0.5.x releases of the MeetingSDK are likely to be unstable and change frequently.

If you are looking to write an application that interfaces with Visionable's V2 architecture, you should remain with v0.4 of the MeetingSDK.

We expect that stability will be established against the V3 architecture with v0.6 of the SDK due when the V3 architecture officially goes into production.

OVERARCHING CHANGES (this text present for ALL v0.5.x releases)
Starting with v0.5, 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 (this text present for ALL v0.5.x releases)
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) -> ())
```

See previous release notes in the v0.5.x series for API changes that were new in previous releases. The rest of these release notes pertain only to the v0.5.3 release.

API CHANGES

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.)

CHANGES/FIXES

Additional work has been done in streamlining the threading architecture. There are no visible changes for this in the APIs or delegate methods. However, at the C++ level, the underlying model object storage has been converted to use C++11 smart pointers (shared_ptr) which should improve any edge cases where data corruption may have occurred.

KNOWN ISSUES

iOS Screen sharing works, but additional work needs to be done when detecting that the user has stopped sharing from the operating system.

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)