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

NOTE:
Version 0.6 of the Visionable Android 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 (this text present for ALL v0.5.x releases)
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 static void initializeMeetingWithToken(
        String token,
        String server,
        String uuid,
        IInitializeMeetingCompleteCallback initDoneCallback
)
```

This function still takes a `uuid` and a `server` name but now also takes a `token` parameter that is either `NULL` 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 joinMeeting (same call for either V2 or V3 servers):

```
public static int joinMeeting(
        String name,
        String userUUID,
        IJoinMeetingCompleteCallback joinDoneCallback)
```

The `initializeMeetingWithToken` function caches the `server` name you are connecting to, the `uuid` for the meeting and the MJWT needed to connect to the meeting. When calling `joinMeeting`, you only need to pass the `name` of the user to be shown in the meeting as well as the optional `userUUID` to be associated with this user (pass an empty string to have the SDK generate a UUID for this user). This behavior is slightly different than the APIs in the MeetingSDK for non-Android platforms, and the Android MeetingSDK will likely be refactored in the future to behave in a manner similar to other platforms.

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 are the same as they were for the v0.4 release of the Android MeetingSDK.

API CHANGES (since the last v0.4 release)

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

```
public static Participant getLocalParticipant()
```

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)

```
public static ArrayList<Participant> getParticipants()
```

The following additional accessor functions have been defined in the MeetingSDK class:

```
public static Participant getParticipantByVideoStreamId(String streamId)
public static Participant getParticipantByAudioStreamId(String streamId)
public static VideoInfo getVideoInfoByStreamId(String streamId)
```

`ILogCallback` is no longer used;
`INotificationCallback` now has the following method:

```
void onLogMessage(int level, String logmessage);
```

If you call the following new API in the `MeetingSDK` class with a `true` parameter:

```
public static void enableInlineAudioVideoLogs(final boolean enable)
```

all log messages from IGAudio and IGVideo will be sent to the `onLogMessage` `INotification` callback as they happen.

INotificationCallback now has the following new methods:

```
void videoStreamBufferReady(String streamId, String pixelBuffer);
```

Called when a new video stream is ready to receive frames from CoreMeeting.   This should be received *after* the corresponding videoViewCreated callback.

```
void videoFrameReady(byte[] frameData, int len);
```

Should be called whenever a new frame comes in from CoreMeeting. It is currently not specified correctly and will not get called.   The v0.5 version of the MeetingSDK will render the incoming frame on the surface associated with the VideoView object; so this call is likely not needed at the moment anyway.

The VideoInfo structure has the following changes:

- The `active` field is now a `Bool`
- The `local` field is now a `Bool`
- The `width` field is now an `Int`
- The `height` field is now an `Int`
- The `layout` field is now an `Int`

The following calls have been changed/added:

```
public static void getVideoDevices(
        ArrayList<String> devices,
        ArrayList<String> screens)
```

Pass in an empty `ArrayList<String>` type for `device` and `screens` parameters and the SDK will fill them in with names of devices that correspond to cameras and named of devices that correspond to screen shares.

```
public static String getAudioInputDevices(ArrayList<String> devices)
public static String getAudioOutputDevices(ArrayList<String> devices)
```

These APIs also now take an empty `ArrayList<String>` type that will be filled in with names of devices found for audio input and audio output devices respectively. The `String` return type will represent what the operating system thinks is the preferred device. Keep in mind that on mobile devices such as Android, you will only ever receive “Default device” as the list of valid input or output devices, and the “preferred device” will also be “Default device”.

```
public static void setDelegate(INotificationCallback delegate)
```

You must now use the `setDelegate` SDK method to notify the SDK of the instance of `INotificationCallback` you are using to receive notifications from the SDK.

To retrieve a `Participant` object based on a known user’s UUID, this function is now available:

```
public static Participant findParticipantByUUID(final String uuid)
```

When setting the delegate with the `MeetingSDK` singleton’s `setDelegate` method, you can now provide a `Looper` that all SDK callbacks will be executed on. A new, overloaded `setDelegate` method has been provided to allow specification of this argument. The original version is still available which, if used, will cause the SDK callbacks to be executed on whatever thread the underlying C++ code that invokes them is on. Additionally, you may now pass NULL for the delegate to tell the `MeetingSDK` singleton that no delegate is available.

```
public static void setDelegate(@Nullable INotificationCallback delegate,
final Looper looper)

public static void setDelegate(@Nullable INotificationCallback delegate)
```

Routines previously available in the `VMeeting` class to retrieve participants based on audio and video stream IDs have been removed. Use new APIs introduced in v0.5 in the `MeetingSDK` singleton for these purposes.

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 `INotificationCallback` method `inputMeterChanged` calls will occur. When audio output preview is enabled, the `INotificationCallback` method `outputMeterChanged` calls will occur.

```
public static boolean enableAudioInputPreview(String device)
```

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. 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 static boolean disableAudioInputPreview(String device)
```

Stop audio input preview events from the device specified which should be one of the strings returned by the `getAudioInputDevices` API call. For Android, 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 static boolean enableAudioOutputPreview(String device,String
soundPath)
```

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 Android, 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. This is a bit tricky on Android devices. Resources (such as .wav files) placed in either the app’s assets directory or in the res directory cannot generally be accessed via an absolute path. One solution would be to have your application access the resources and then copy them (programmatically) into a directory (such as the one returned by `context.getFilesDir()`) which you *can* access them via an absolute file path.

```
public static boolean disableAudioOutputPreview(String device)
```

Stop audio output preview events from the device specified which should be one of the strings returned by the `getAudioOutputDevices` API call. For Android, 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 static boolean enableVideoPreview(String deviceName, String
resolution)
```

Starts a video preview session with the specified camera. The camera `deviceName` parameter should be one of the values returned by `getVideoDevices`. The `resolution` parameter is one of the standard supported resolutions (same ones used for the `enableVideoStream` API call). When the video preview has started, you will be notified via a new `INotificationCallback` routine, documented below.

Note: While, in theory, it should be possible to begin a video preview session while you are already in a meeting, the behavior or attempting to preview the back camera while the front camera is being used for a meeting (or vice versa) has not been verified.

```
public static boolean disableVideoPreview(String deviceName)
```

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

The following new `INotificationCallback` method has been added:

```
default void previewVideoViewCreated(VideoView videoView) {}
```

Called whenever a new preview video view is ready to be used. The `VideoView` object passed to this interface method is one that can be embedded in your user interface.

Additionally, all interface methods in `INotificationCallback` have been given default definitions (which do nothing) which means that you only need to implement the methods in `INotificationCallback` that you are interested in.

```
public static boolean disableVideoPreview(String deviceName)
```

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

Video codec constants have changed:

```
public final static String H_261_Standard = "H.261 Standard";
public final static String SMALL_NAME = "SMALL";
public final static String MEDIUM_NAME = "MEDIUM";
public final static String LARGE_NAME = "LARGE";
public final static String HD1_NAME = "HD1";
public final static String HD2_NAME = "HD2";
public final static String HD3_NAME = "HD3";
public final static String HIGH_4K_NAME = "4K HIGH";
public final static String HD = HD1_NAME;
public final static String SD = LARGE_NAME;
```

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

```
public static int joinMeeting(
        String name,
        String userUUID,
        IJoinMeetingCompleteCallback joinDoneCallback)
```

When connecting to a V3 meeting, the `userUUID` parameter is ignored. In the context of a V3 connection, the user's unique UUID is pulled from the MJWT passed in. If a guest MJWT is requested, it will have it's own unique user UUID associated with it, and that unique UUID will be used for the guest user . 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.

A new INotificationCallback has been added to notify you of when an active preview video stream is updated (usually due to the rotation of a tablet while the preview is active):

```
default void previewVideoUpdated(String streamId) {};
```

CHANGES/FIXES

Changes since the v0.5.19 release:

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

Fixed a crash situation that could occur when starting or stopping Android screen sharing.

Fixed an uncommon crash situation that could occur when a remote user leaves the meeting.

KNOWN ISSUES

The newly added `previewVideoUpdated` callback is not active called at this time.   While you can provide an implementation of it, your implementation will not be called until the next SDK update.

While the ability to specify a dedicated Looper upon which all delegate methods are invoked, the current Android MeetingSDK does not attempt to create a dedicated Looper if one is not specified.   This will result in all delegate method calls being made on the same thread being used to parse low-level audio and video events coming from our audio/video engine.

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