WINDOWS MEETINGSDK

The Visionable Windows MeetingSDK package brings our Windows offering up to par with MeetingSDK distributions on other platforms. It is distributed as a single .zip file containing the C++ header files as well as the MeetingSDK DLL and all dependent DLLs. There DLL has 3 singletons and 2 delegate classes which are documented here:

- MeetingSDK
- ModeratorSDK
- VisionableAPI
- MeetingSDKDelegate
- ModeratorSDKDelegate

MEETINGSDK API CALLS

```
static MeetingSDK* sharedInstance()
```

The shared instance of the MeetingSDK singleton that should be used when interacting with MeetingSDK. For example, to make the call to exit a meeting, you'd use the following code:

```
MeetingSDK::sharedInstance()->exitMeeting();
```

```
void setDelegate(MeetingSDKDelegate* delegate)
```

Sets the delegate object for the `MeetingSDK` singleton.

Parameters:

- Delegate – A pointer to an instance of a class derived from `MeetingSDKDelegate` that implements delegate methods for the purposes of receiving feedback/messages from the SDK.

```
void joinMeeting(const std::string& server, const std::string&
     uuid, const std::string& key, const std::string& userUUID,
     const std::string& name, std::function<void(bool,
     std::string)> completion);
```

Join a meeting using the V2 protocols.

Parameters:

- server – the DNS name of the server you wish to connect to
- uuid – The meeting UUID of the meeting you wish to join
- key – The 44 byte encryption key obtained from a call to initializeMeeting (VisionableAPI)
- userUUID – The UUID of the user joining this meeting (in V2 protocol, usually an email address)
- name – The "display name" this user wishes to use in the meeting
- completion – A function pointer/lambda of code to be executed when the join process is completed. Parameters to this function are a Boolean flag indicating

whether or not the join process was initialized successfully and a string containing the device-specific UUID generated for this user on the device being used.

```
void joinMeetingWithToken(const std::string& server, const
     std::string& uuid, const std::string& token, const
     std::string& userUUID, const std::string& name,
     std::function<void(bool, std::string)> completion);
```

Join a meeting using the V3 protocols. Assumes you already know the authenticated user's UUID OR you are attempting to join as a guest user.

Parameters:

- server – the DNS name of the server you wish to connect to
- uuid – The meeting UUID of the meeting you wish to join
- token – An MJWT (guest or authenticated) obtained from a call to initializeMeeting (VisionableAPI)
- userUUID – The uuid of the user joining the meeting OR an empty string for a guest user.
- name – The "display name" this user wishes to use in the meeting
- completion -- function point/lambda of code to be executed when the join process is completed. Parameters to this function are a Boolean flag indicating whether or not the join process was initialized successfully and a string containing the device-specific UUID generated for this user on the device being used.

```
void joinMeetingWithTokenAndJWT(const std::string& server, const
     std::string& uuid, const std::string& token, const
     std::string& jwt, const std::string& name,
     std::function<void(bool, std::string)> completion);
```

Join a meeting using the V3 protocols. Assumes you have access to the authenticated user's JWT and want the SDK to pull out the corresponding user UUID when joining. This API cannot be used to join as a guest.

Parameters:

- server – the DNS name of the server you wish to connect to
- uuid – The meeting UUID of the meeting you wish to join
- token – An MJWT (guest or authenticated) obtained from a call to initializeMeeting (VisionableAPI)
- jwt – The same JWT used to obtain the MJWT (via a call to initializeMeeting).
- name – The "display name" this user wishes to use in the meeting

- completion -- function point/lambda of code to be executed when the join process is completed. Parameters to this function are a Boolean flag indicating whether or not the join process was initialized successfully and a string containing the device-specific UUID generated for this user on the device being used.

```
bool enableAudioInput(const std::string& device);
```

Enable the specified audio device for input into the meeting.

Parameters:
- device -- a device name returned from a call to getAudioInputDevices.

Returns:
- true if the enable operation completed successfully, false otherwise.

```
bool disableAudioInput(const std::string& device);
```

Disables the specified audio device for input into the meeting.

Parameters:
- device -- a device name returned from a call to `getAudioInputDevices`.

Returns:
- true if the disable operation completed successfully, false otherwise.

```
bool enableAudioOutput(const std::string& device);
```

Enable the specified audio device for output from the meeting.

Parameters:
- device -- a device name returned from a call to `getAudioOutputDevices`.

Returns:
- true if the enable operation completed successfully, false otherwise.

```
bool disableAudioOutput(const std::string& device);
```

Disables the specified audio device from receiving output from the meeting.

Parameters:

- device -- a device name returned from a call to `getAudioOutputDevices`.

Returns:

- true if the disable operation completed successfully, false otherwise.

```
bool enableAudioInputPreview(const std::string& device);
```

Enable the specified audio device for input preview. If successfully enabled, your MeetingSDKDelegate will begin receiving `inputMeter` callbacks relating to the device being previewed.

Parameters:

- device -- a device name returned from a call to `getAudioInputDevices`.

Returns:

- true if the enable operation completed successfully, false otherwise.

```
bool disableAudioInputPreview(const std::string& device);
```

Disables preview for the specified audio device.

Parameters:

- device -- a device name returned from a call to `getAudioInputDevices`.

Returns:

- true if the disable operation completed successfully, false otherwise.

```
bool enableAudioOutputPreview(const std::string& device,
     const std::string& soundURL);
```

Enable the specified audio device for output preview.

Parameters:

- device -- a device name returned from a call to `getAudioOutputDevices`.

- soundURL – A file-based URL that identifies a .wav file that can be played on the device being previewed.

Returns:
- true if the enable operation completed successfully, false otherwise.

---

bool disableAudioOutputPreview(const std::string& device);

Disables preview for the specified audio device.

Parameters:
- device -- a device name returned from a call to `getAudioOutputDevices`.

Returns:
- true if the disable operation completed successfully, false otherwise.

```
bool setAudioStreamVolume(const std::string& streamId, uint8_t
volume);
```

Sets the volume for an individual remote user in the meeting.

Parameters:
- streamId – The (audio) stream id of the user whose volume you'd like to change
- volume – An integer from 0-100 reflecting the volume level

Returns:
- true if the set volume operation was successfully initiated, false if not

---

```
bool setAudioInputVolume(const std::string& device,
     uint8_t volume);
```

Sets the input volume for a specified audio input device.

Parameters:
- device – The name of the device you wish to set the input level for
- volume – An integer from 0-100 reflecting the input level

Returns:

- true if the set input volume operation was successfully initiated, false if not

---

```
bool setAudioOutputVolume(const std::string& device,
     uint8_t volume);
```

Sets the output volume for a specified audio output device.

Parameters:

- device – The name of the device you wish to set the output level for
- volume – An integer from 0-100 reflecting the output level

Returns:

- true if the set output volume operation was successfully initiated, false if not

---

```
bool enableVideoCapture(const std::string& camera,
     const std::string& mode,
     bool enableBlurring = false);
```

Enables the specified video device to send video into the meeting.

Parameters:

- camera – The name of the camera device to enable, should be one of the devices returned by a call to `getVideoDevices`().
- mode – The "send resolution" to use. Should be a value returned by `getSupportedVideoSendResolutions`().
- enableBlurring – Send "true" if you would like the background blurred or "false" if not.

Returns:

- true if the set enable video capture operation was successfully initiated, false if not

---

```
bool disableVideoCapture(const std::string& camera);
```

Disables the specified video device from sending video into the meeting.

Parameters:

- camera – The name of the camera device to disable, should be one of the devices returned by a call to `getVideoDevices`().

Returns:

- true if the set disable video capture operation was successfully initiated, false if not

---

```
bool enableVideoPreview(const std::string& camera,
     const std::string& mode, bool enableBlurring = false);
```

Enables the specified video for preview. The allows your application to receive video frames from the device without being in a meeting. See MeetingSDKDelegate documentation on how to receive frames.

Parameters:

- camera – The name of the camera device to enable preview for, should be one of the devices returned by a call to `getVideoDevices`().
- mode – The "send resolution" to use. Should be a value returned by `getSupportedVideoSendResolutions`().
- enableBlurring – Send "true" if you would like the background blurred or "false" if not.

Returns:

- true if the set enable video preview operation was successfully initiated, false if not

---

```
bool disableVideoPreview(const std::string& camera);
```

Disables the preview for the specified video device.

Parameters:

- camera – The name of the camera device to disable preview for, should be one of the devices returned by a call to `getVideoDevices`().

Returns:

- true if the set disable video preview operation was successfully initiated, false if not

---

```
bool enableWindowSharing(const std::string& windowId,
     const std::string& mode);
```

Shares the specified window into the meeting.

Parameters:

- windowId – The window id of the window you wish to share into the meeting. Can be obtained with a call to getWindowList().
- mode – The screen sharing mode you'd like to use when sharing this window. Should be one of the following values:
  - 4K LOW
  - LOW SCREEN
  - MED SCREEN
  - HIGH SCREEN
  - BEST SCREEN
  - LOSSLESS1
  - LOSSLESS2
  - LOSSLESS3

Returns:

- true if the set enable window sharing operation was successfully initiated, false if not

---

```
bool disableWindowSharing(const std::string& windowId);
```

Disables the preview for the specified video device.

Parameters:

- windowId – The window id of the window you wish to stop sharing into the meeting. Can be obtained with a call to getWindowList().

Returns:

- true if the set disable window sharing operation was successfully initiated, false if not

---

```
bool enableNetworkVideo(const std::string& url,
     const std::string& mode, const std::string& name);
```

Enables video from a network (IP) based camera to be sent into the meeting.

Parameters:

- url – The url used to connect to the network video source

- mode – The screen sharing mode you'd like to use when sharing this network feed. Should be one of the following values:
    - 4K LOW
    - LOW SCREEN
    - MED SCREEN
    - HIGH SCREEN
    - BEST SCREEN
    - LOSSLESS1
    - LOSSLESS2
    - LOSSLESS3

Returns:

- true if the share network video operation was successfully initiated, false if not

---

```
bool disableNetworkVideo(const std::string& url);
```

Disables an active network video feed

Parameters:

- url – The url used to connect to the network video source

Returns:

- true if the disable network video operation was successfully initiated, false if not

---

```
bool disableVideoStream(const std::string& streamId);
```

Stop receiving video frames for a given video stream in the current meeting.

Parameters:

- streamId – The stream id of the video stream to stop receiving frames for.

Returns:

- true if the disable video stream operation was successfully initiated, false if not

---

```
bool enableVideoStream(const std::string& streamId);
```

Begin receiving video frames for a given video stream in the current meeting. Currently this results in a low-level feed of frames directly to your application (See the MeetingSDKDelegate class reference for more information). Frames will be formatted (by default) in the BGR888 colorspace. In the future, an overloaded version of this API which allows you to request that the SDK hand you a user interface object that can be inserted directly into your view hierarchy will be provided.

Parameters:

- streamId – The stream id of the video stream to begin receiving frames for.

Returns:

- true if the enable video stream operation was successfully initiated, false if not

---

```
bool enableVideoStream(const std::string& streamId,
     const std::string& colorspace);
```

Begin receiving video frames for a given video stream in the current meeting. Frames will be formatted in the specified color space. Currently this results in a low-level feed of frames directly to your application (See the MeetingSDKDelegate class reference for more information). In the future, an overloaded version of this API which allows you to request that the SDK hand you a user interface object that can be inserted directly into your view hierarchy will be provided.

Parameters:

- streamId – The stream id of the video stream to begin receiving frames for.
- Colorspace – An alternate colorspace to format frames in.

Returns:

- true if the enable video stream operation was successfully initiated, false if not

---

```
void pauseVideoFrameProcessing(const std::string& streamId);
```

Ask the SDK to stop providing video frames for the specified, enabled video stream. This doesn't prevent video frames from being sent to your device, just prevents them from being forwarded to your application.

Parameters:

- streamId – The stream id of the video stream to pause frames for.

```
void resumeVideoFrameProcessing(const std::string& streamId);
```

Ask the SDK to resume providing video frames for the specified, enabled video stream.

Parameters:

- streamId – The stream id of the video stream to resume frames for.

```
void exitMeeting();
```

Disconnect from the current meeting. When disconnecting from a meeting you will not receive delegate callbacks (such as `participantVideoRemoved`) for active video streams in the meeting at the time of disconnect.

```
void enableCombinedLogs(bool enable);
```

Tell the SDK whether or not to keep logs for the audio, video and sdk engines separate or if they should be merged together into one log. By default, logs are cached in separate buffers and only available through API calls to retrieve them.

Parameters:

- enable – Pass “true” if you’d like all logs to be aggregated together in one buffer. Pass “False” (default) to have them kept separate.

```
void enableLogForwarding(bool enable);
```

Instead of buffering log entries, make them available to the application through the MeetingSDKDelegate logMessage callback. If logs are not being combined, this will only capture log entries from CoreMeeting and MeetingSDK (but not the audio/video engines).

Parameters:

- enable – Pass “true” to have log messages forwarded up to the application through MeetingSDKDelegate, pass “false” to leave them buffered.

```
void enableActiveLogging(const std::string& filename);
```

Ask the SDK to store log messages for CoreMeeting/MeetingSDK in the specified filename. Assumes you have called `setLogDirectory` to specify which directly log files should be store in. If combined logs are enabled, the log file will also receive log entries from the audio/video engines.

Parameters:

- filename – The filename to write log entries into.

```
void setTraceLevel(int level);
```

Set level of details logged in all modules (audio, video, CoreMeeting, MeetingSDK). Currently this API is the only way to set the logging level for CoreMeeting and MeetingSDK.

Parameters:

- level – an integer between 0-7 specify the log level. In the future, log level constants will be provided for better code readability.

VALID INPUT AND MEANINGS

0. (none) – No information will be logged
1. (err) – Only errors logged
2. (warn) – Add abnormal behaviors
3. (info) – Add normal behaviors
4. (dbg1) – Basic debugging information
5. (dbg2) – Extra debugging information
6. (dbg3) – specific debugging information
7. (dbg4) – extremely verbose debugging information

```
void audioTraceLevel(int level);
```

Set level of details logged for the audio engine only.

Parameters:

- level – an integer between 0-7 specify the log level. In the future, log level constants will be provided for better code readability.

VALID INPUT AND MEANINGS

8. (none) – No information will be logged
9. (err) – Only errors logged
10. (warn) – Add abnormal behaviors
11. (info) – Add normal behaviors
12. (dbg1) – Basic debugging information
13. (dbg2) – Extra debugging information
14. (dbg3) – specific debugging information
15. (dbg4) – extremely verbose debugging information

```
void videoTraceLevel(int level);
```

Set level of details logged for the video engine only.

Parameters:

- level – an integer between 0-7 specify the log level. In the future, log level constants will be provided for better code readability.

VALID INPUT AND MEANINGS

16. (none) – No information will be logged
17. (err) – Only errors logged
18. (warn) – Add abnormal behaviors
19. (info) – Add normal behaviors
20. (dbg1) – Basic debugging information
21. (dbg2) – Extra debugging information
22. (dbg3) – specific debugging information
23. (dbg4) – extremely verbose debugging information

```
void videoTraceOutputHistory(const std::string& filename);
```

Write buffered log entries for the video engine to the file specified. This should be an absolute file path and does not depend on the directory specified in a call to `setLogDirectory`(). The file will be filled with all currently buffered entries, but will not stay connected to the logging process.

Parameters:

- filename – The absolute path to the file to be written.

```
void audioTraceOutputHistory(const std::string& filename);
```

Write buffered log entries for the audio engine to the file specified. This should be an absolute file path and does not depend on the directory specified in a call to `setLogDirectory`(). The file will be filled with all currently buffered entries, but will not stay connected to the logging process.

Parameters:

- filename – The absolute path to the file to be written.

```
void coreMeetingTraceOutputHistory(const std::string& filename);
```

Write buffered log entries for the CoreMeeting/MeetingSDK layer to the file specified. This should be an absolute file path and does not depend on the directory specified in a call to `setLogDirectory`(). The file will be filled with all currently buffered entries, but will not stay connected to the logging process.

Parameters:

- filename – The absolute path to the file to be written.

```
void getVideoDevices(std::vector<std::string>& devices,
     std::vector<std::string>& screens);
```

Returns a vector of device names corresponding to available Desktop (screen) shares and camera devices on the local machine.

Parameters:

- devices – a pass-by-reference vector of strings that will be filled with the names of all camera devices available to be used in the meeting.
- Screens – a pass-by-reference vector of strings that will be filled with the names of all screen/desktop devices available to be shared into the meeting.

```
void getAudioInputDevices(std::vector<std::string>& devices,
     std::string& preferred);
```

Returns a vector of device names corresponding to available audio input devices on the local machine. Also returns the name of the device the SDK believes to be the user's preferred device.

Parameters:

- devices – a pass-by-reference vector of strings that will be filled with the names of all audio input available to be used in the meeting.
- preferred – The name of the device the SDK believes to be the user's preferred audio input device.

```
void getAudioOutputDevices(std::vector<std::string>& devices,
     std::string& preferred);
```

Returns a vector of device names corresponding to available audio output devices on the local machine. Also returns the name of the device the SDK believes to be the user's preferred device.

Parameters:

- devices – a pass-by-reference vector of strings that will be filled with the names of all audio output available to be used in the meeting.
- preferred – The name of the device the SDK believes to be the user's preferred audio output device.

```
void getSupportedVideoSendResolutions(const std::string& device,
     std::vector<std::string>& resolutions);
```

Given the specified video device, returns a vector of valid screen resolutions for that device.

Parameters:

- device – The name of the video device to get send resolutions for
- resolutions – a pass-by-reference vector of strings to be filled with each supported video send resolution.

```
int getWindowList(std::list<std::pair<uint64_t, std::string>>&
      windowList);
```

Retrieve a list of available windows for sharing into the meeting. An application can use the information returned to present users with a list of window names to choose from (as well as the integer identifier needed to enable the sharing of that window into the meeting)

Parameters:

- windowList – A pass-by-reference list of C++ pair structures where each pair consists of a 64-bit identifier (integer) for the window and a string containing the name of that window.

```
bool getWindowThumbnail(const std::string& id,
     const std::string& outputFormat,
     const std::string& destPath, int width, int height,
     uint8_t*& pThumbBuffer, uint32_t& size);
```

Retrieve a thumbnail (of a specific size) for a particular window. This can be used when implementing a sharing UI.

Parameters:

- id – A string representation of the 64-bit integer window id returned by a call to getWindowList()
- outputFormat – the format you wish the thumbnail to be stored in
- destPath – If non-empty, a directory path into which the thumbnail will be written
- width – Width you'd like the thumbnail to have
- height – Height you'd like the thumbnail to have
- pThumbBuffer – pointer to a memory buffer where the image can be stored (pass null if writing to a file)
- size – The size of the buffer written to in pThumbBuffer

A thumbnail retrieved using this API call must be freed with `releaseWindowThumbnail`.

```
bool releaseWindowThumbnail(const std::string& id,
     uint8_t*& pThumbBuffer);
```

Release memory returned by a previous call to `getWindowThumbnail`.

Parameters:

- id – A string representation of the 64-bit integer window id returned by a call to `getWindowList`()
- pThumbBuffer – The previously allocated buffer containing the thumbnail.

Returns:

- "true" if the thumbnail was successfully released, "false" if a problem occurred.

```
const void* playSound(const char* soundData, size_t size);
```

Plays the sound data (WAV format) passed in. Any sound played through this mechanism will be included in the echo-cancellation logic.

Parameters:

- soundData – a byte array containing the WAV sound data
- size – length of the data passed in the soundData parameter

Returns:

- a pointer which serves as an identifier for this sound to be used with other calls.

```
bool stopSound(const char* soundData);
```

Stops playing the sound previous played with a call to `playSound`().

Parameters:

- `soundData` – The byte array passed to `playSound`() to play the sound you now wish to stop.

```
bool isScreenShare(const std::string& codec);
```

When passed the codec associated with a video stream, will identify if that stream is a screen share or not.

Parameters:

- codec – the name of the codec associated with the video stream

Returns:

- “true” if the stream represents a screen share, “false” if not.

```
std::string getLastError();
```

Retrieves the last error stored when using any API in the MeetingSDK. If a given API call returns “false” to indicator some sort of failure, use this API to get a string representation of what went wrong.

Returns:

- An error string describing the last problem encountered by the SDK

```
void getLocalParticipant(Participant& participant);
```

Returns the `Participant` object corresponding to the local user for the current meeting.

Returns:

- The local user’s `Participant` object

```
void getParticipantIds(std::vector<std::string>& ids);
```

Retrieve participant IDs for all users in the current meeting.

Returns:

- A vector of strings each of which corresponds to the id of each user in the current meeting.

```
bool findVideoInfo(const std::string& streamId,
     VideoInfo& videoInfo);
```

Retrieve a `VideoInfo` object corresponding to the stream id specified.

Parameters:

- streamId – The stream id of the video stream you wish to obtain a VideoInfo object for.
- videoInfo – A pass-by-reference VideoInfo object that will be "filled in" with information about the specified video stream.

Returns:

- "true" if a VideoInfo object was found for the specified stream id, "false" if not.

```
bool findParticipantByVideoStreamId(const std::string& streamId,
     Participant& participant);
```

Retrieve a `Participant` object for the participant who "owns" the video feed associated with `streamId`.

Parameters:

- streamId – The stream id of a video feed belonging to the participant you'd like to find.
- participant – A pass-by-reference value to be filled in with participant info

Returns:

- "true" if a `Participant` record was found for the participant who owns the specified stream id, "false" if not.

```
bool findParticipantByAudioStreamId(const std::string& streamId,
     Participant& participant);
```

Retrieve a `Participant` object for the participant who "owns" the audio feed associated with `streamId`.

Parameters:

- streamId – The stream id of a audio feed belonging to the participant you'd like to find.
- participant – A pass-by-reference value to be filled in with participant info

Returns:

- "true" if a `Participant` record was found for the participant who owns the specified stream id, "false" if not.

```
bool findParticipantByUUID(const std::string& uuid,
     Participant& participant);
```

Retrieve a `Participant` object for the with the specified user UUID.

Parameters:

- uuid – The user UUID of the participant you'd like to find.
- participant – A pass-by-reference value to be filled in with participant info

Returns:

- "true" if a `Participant` record was found for the specified user UUID, "false" if not.

```
bool setLogDirectory(const std::string& path);
```

Sets the directory in which CoreMeeting/MeetingSDK logs will be written if requested. If this directory doesn't exist, it will be created if possible.

Parameters:

- path – An absolute path to the directory that either exists or which should be created to store log files.

Returns:

- "true" if the directory could be created/established as the logging location, "false" if some error occurred.

```
bool deleteLogFile(const std::string& fileName);
```

Delete the specified log file.

Parameters:

- filename – Name of the file to be deleted from the directory specified by a call to `setLoggingDirectory`()

Returns:

- “true” if the file was deleted, “false” if an error occurred.

```
bool deleteAllLogFiles();
```

Delete all log files in the logging directory. A log file is identified with a `.v1log` extension.

Returns:

- “true” if log files could be deleted, “false” if a problem occurred.

```
bool resetCurrentLogFile();
```

Empties the current log file (the file set with a call to `enableActiveLogging`()).
Returns:

- “true” if the current log file was able to be emptied, “false” if an error occurred.

```
bool trimCurrentLogFile(int numBytes);
```

CURRENTLY NOT IMPLEMENTED, PLACEHOLDER API

```
bool getLogFiles(std::vector<std::string>& fileNames);
```

Return a vector of log file names in the logging directory.

Parameters:

- fileNames – a pass-by-reference vector of strings that is filled in with names of all logs (files with a `.v1log` extension) in the current logging directory.
-

```
bool flushCurrentLogFile();
```

Write any buffered log entries out to the current log file.

Returns:

- “true” if buffered log entries were successfully flushed, “false” if some error occurred.

```
void getTimeZone(std::string& timeZone);
```

Returns a string representation of the current time zone the user's device is in.

Parameters:

- timeZone – a pass-by-reference string filled in with the current time zone.

MODERATORSDK API CALLS

```
static ModeratorSDK* sharedInstance()
```

The shared instance of the ModeratorSDK singleton that should be used when interacting with ModeratorSDK. For example, to make the call that sends an updated list of your current devices to the RTN service, you'd use the following code:

```
ModeratorSDK::sharedInstance()->sendDeviceListUpdate();
```

```
void setDelegate(ModeratorSDKDelegate* delegate)
```

Sets the delegate object for the `ModeratorSDK` singleton.

Parameters:

- Delegate – A pointer to an instance of a class derived from `ModeratorSDKDelegate` that implements delegate methods for the purposes of receiving feedback/messages from the SDK.

```
bool isLocalUser(std::string userUUID)
```

Given the userUUID specified, check to see if it belongs to the local user.

Parameters:

- userUUID – The user UUID to check against the local user

Returns:

- "true" if the specified uuid belongs to the local user, "false" if not.

```
void connectWebSocket(std::string meetingUUID,
     std::string generatedUUID,
     std::string msgServer, std::string userUUID,
     std::string mjwt)
```

Connect to the RTN service. Applications don't need to call this method directly; it will be called automatically by MeetingSDK when a meeting is joined.

Parameters:

- meetingUUID – The meeting UUID of the meeting just joined
- generatedUUID – The generated UUID for this user's device (for the current meeting)
- msgServer – The server where the RTN service is running
- userUUID – The user's UUID (as taken from their JWT)
- mjwt – The MJWT for this user for this meeting

```
void sendMessage(std::string destination, std::string message)
```

Sends a message to a specific user via the RTN service.

Parameters:

- destination -- The UUID of the user to send a message to. This is obtained from the Participant record of the desired user.
- Message – The JSON message to send to the specified user.

```
bool sendPTZCommand(std::string user, std::string device,
     std::string command)
```

Sends a PTZ command to the specified user in the current meeting.

Parameters:

- user – The UUID of the user to send a message to. This is obtained from the `Participant` record of the desired user.
- device – The name of the device on the desired user's machine to send the command to.
- command – The command to send to the target device. A list of valid commands is forthcoming.

Returns:

- "true" if the PTZ command could be sent, "false" if not.

```
void setRemotePTZAllowed(bool allowed)
```

Sets whether or not this user will allow remote users to control a local PTZ camera.

Parameters:

- Allowed – A Boolean indicating this user's preference for remote PTZ control. Pass "true" to allow remote users to control local cameras, "false" if not.

```
bool getRemotePTZAllowed()
```

Retrieve the user's preference for allowing remote PTZ control

Returns:

- "true" if the user is allowing remote users to control a local camera, "false" if not.

```
void sendDeviceListUpdate()
```

Send an update on the local devices available to the current meeting to the RTN service. Applications do not need to call this directly; it is called by MeetingSDK whenever a device is enabled/disabled/added.

```
void closeSession()
```

Closes the current web socket connection to the RTN. Applications do not need to call this directly.

VISIONABLEAPI API CALLS

```
static VisionableAPI* sharedInstance()
```

The shared instance of the VisionableAPI singleton that should be used when interacting with VisionableAPI. For example, to make the call that attempts to authenticate a user, you would use:

```
VisionableAPI::sharedInstance()->authenticate( /* params */);
```

```
void authenticate(std::string server, std::string id,
     std::string password,
     std::function<void(std::string)> completion);
```

Used to authenticate a user via Cognito. As such, only available for V3/later connections.

Parameters:

- server – The API server to authenticate to
- id – The user's ID
- password – The user's password
- completion – A lambda expression the is called when authentication completes. It takes a string that is either empty (failure) or populated with the user's JWT token.

```
bool initializeMeeting(std::string server,
     std::string meetingUUID,
     std::function<void(bool, std::string)> completion);
```

Used to retrieve a "meeting key" for a V2 meeting. The SDK does not support authentication for older, V2 servers, so the assumption is that if you will be joining a V2 meeting, it will be as an anonymous user.

Parameters:

- server -- The server to connect to that is hosting the meeting
- meetingUUID – The UUID of the meeting you wish to join
- completion – A callback (function pointer or lambda) that represents a function which returns void and takes two parameters: (1) a bool indicating whether or not the meeting was successfully initialized and (2) the meeting key to be used for this meeting.

```
bool initializeMeeting(std::string server,
     std::string meetingUUID, std::string token,
     std::function<void(bool, std::string)> completion);
```

Used to retrieve an MJWT for a V3 or later meeting. The MJWT returned via the callback function will be either an anonymous or authenticated MJWT depeneding on the value of the token parameter passed in.

Parameters:

- server – The “base” server name you wish to connect to (without a -ucs/-api/-msg extension)
- meetingUUID – The UUID of the meeting you wish to obtain an MJWT for
- token – An empty string for anonymous joining or the JWT of the user you wish to join the meeting as
- completion – A callback (function pointer or lambda) that represents a function which returns void and takes two parameters: (1) a bool indicating whether or not the. Meeting was successfully initialized and (2) the MJWT for this user and this meeting.

DATA STRUCTURES

The `VideoInfo` data structure contains information about a given video stream. This structure is a "front" structure for data stored down at the CoreMeeting layer. As such, only the `streamId` field is stored in this structure, all other properties are accessed via public accessor functions and retrieve data from the CoreMeeting layer.

```
class DLLEXPORT VideoInfo {
public:
    VideoInfo();
    VideoInfo(std::string streamId);
    std::string site();
    std::string name();
    std::string codecName();
    bool local();
    bool active();
    bool ptzStatus();
    uint8_t layout();
    uint32_t width();
    uint32_t height();
public:
    std::string streamId;
};
```

Fields:

- streamId – The stream id of the video stream this structure represents
- site() – Retrieve the name of the site associated with this video stream. This is usually the user's display name.
- name() – Retrieve the name of the device associated with this video stream
- codecName() – Retrieve the codec being used with this video stream
- local() – Returns true if this stream belongs to the local user, returns false otherwise.
- active() – Returns true if this stream is considered active, false if not.
- ptzStatus() – Returns true if this camera can be controlled remotely via PTZ commands.
- layout() – reserved
- width() – Returns the width (in pixels) of this video stream
- height() – Returned the height (in pixels) of this video stream

The `AudioInfo` data structure contains information about a given audio stream. This structure is a "front" structure for data stored down at the CoreMeeting layer. As such, only the `streamId` field is stored in this structure, all other properties are accessed via public accessor functions and retrieve data from the CoreMeeting layer.

```
class DLLEXPORT AudioInfo {
public:
    AudioInfo(std::string streamId);

    std::string site();
public:
    std::string streamId;
};
```

Fields:

- streamId – The stream id of the audio stream this structure represents
- site() – Retrieve the name of the site associated with the audio stream. This is usually the user's display name.

The `Participant` data structure contains information about a particular user in the meeting. This structure is a "front" structure for data stored down at the CoreMeeting layer. As such, only the `userUUID` field is stored in this structure, all other properties are accessed via public accessor functions and retrieve data from the CoreMeeting layer.

```
class DLLEXPORT Participant {
public:
    Participant(std::string userUUID);
    AudioInfo audioInfo();
    std::map<std::string, VideoInfo> videoInfo();
    std::string displayName();
    bool isLocal();

public:
    std::string userUUID;
};
```

Fields:

- userUUID – The UUID associated with this user. This normally is a combination of the user's UUID as obtained from their JWT and a unique UUID generated to identify their device.
- audioInfo() – Retrieved the AudioInfo structure associated with this user.

- videoInfo() – Retrieved a map containing a key-value pair for each video stream this user is sending into the meeting. The key-value pair consists of the video stream id as the key and the corresponding `VideoInfo` structure as the value.
- displayName() – Retrieve the name to display for this user.
- isLocal() – Retrieve whether or not this `Participant` is the local user.

MEETINGSDK DELEGATE

The `MeetingSDKDelegate` class represents an "interface" of member functions that may be called by the SDK to notify the application of certain events. It is defined as an abstract class with default implementations for most methods that do nothing. To receive callbacks in your application, you must derive a new class from the `MeetingSDKDelegate` class and implement the member functions you would like the SDK to call for you. Then call the `MeetingSDK::setDelgate()` member function passing a pointer to the instance of your class. Each member function is listed below:

MeetingSDKDelegate Member Functions:

```
void meetingToken(std::string decodedToken)
```

When joining a V3 or later meeting, this member function will be called and you will be passed the decoded MJWT for the current user in the current meeting (will be a JSON string)

```
void participantAdded(const Participant participant);
```

Called when the SDK detects that there is a new participant in the meeting. The participant argument will contain information in the form of a Participant object. This is a pure virtual member function that *must* be implemented.

```
void participantRemoved(const Participant participant);
```

Called when a participant leaves the current meeting. This callback will normally be received after callbacks noting that video and audio streams have been removed are invoked. This is a pure virtual function that *must* be implemented.

```
void participantAudioAdded(const Participant participant)
```

Called when a new audio stream is detected for the participant passed in the participant argument. You can get information about the audio stream using the `audioStream`() accessor function in the `Participant` object passed.

```
void participantAudioUpdated(const Participant participant)
```

Called when something changed with the audio stream for the given participant. Use the `audioStream`() accessor function in the `Participant` object passed for more information. This is a pure virtual function and *must* be implemented.

```
void participantVideoAdded(const Participant participant,
     std::string streamId)
```

Called when a new video stream is detected in the current meeting. Common practice would be, upon receiving this callback, call MeetingSDK::enableVideoStream() to begin receiving frames.

```
void participantVideoUpdated(const Participant participant,
     std::string streamId)
```

Called when something about the specified video stream has changed. Examine the appropriate `VideoInfo` object accessible in the passed `Participant` object to get more information on what has changed.

```
void participantVideoRemoved(const Participant participant,
     std::string streamId)
```

Called when a remote participant has stopped sending video for a given video device into the meeting. The `streamId` parameter identifies the video stream removed; upon receiving this event no additional frames should come in.

```
void participantVideoRemoteLayoutChanged(const Participant
     participant, std::string streamId)
```

Called when, in a V2 meeting, a new “layout number” has been assigned to a given video stream by a moderator/meeting owner. This mechanism is unique to the Visionable V2 client and is not likely useful for other applications

---

```
void videoStreamBufferReady(std::string streamId,
     void* pixelBuffer)
```

Called when the underlying video engine has allocated a new shared memory buffer to receive frames for this video stream. Once this event is received, you may begin receiving `videoFrameReady` callbacks. There is no need to implement functionality for this member function.

```
void videoFrameReady(std::string streamId, void* pixelBuffer)
```

Called when a new video frame is ready to be rendered. Frame will be in the format (colorspace) specified when enabling the video stream, the `pixelBuffer` parameter contains a pointer to that memory.

```
void videoPreviewReady(std::string streamId, std::string name,
     int width, int height)
```

Called when a video stream associated with a video preview is ready to be enabled. The stream id, information about the preview device's name along with the width and height of the frames to be received are all provided as arguments to this member function. Call `enableVideoStream` to begin receiving frames for the preview.

```
void previewFrameReady(std::string streamId, void* pixelBuffer)
```

Called when a new frame for a preview is ready to be rendered. Frame will be in the format (colorspace) specified when enabling the video stream, the `pixelBuffer` parameter contains a pointer to that memory.

```
virtual void previewVideoUpdated(std::string streamId) {}
```

Called when something about the specified preview video stream has changed. Since no associated `Participant` object is provided, call `MeetingSDK::findVideoInfo()` with the `streamId` passed to this member function to get the corresponding `VideoInfo` object. Then look for what has changed!

```
void videoError(std::string errorName, std::string errorDesc,
     bool isFatal)
```

Called when the underlying video engine sends an error message to the SDK. Will contain a brief error name, a full description as well as a Boolean indicating whether or not the error is considered a fatal error.

```
void audioError(std::string errorName, std::string errorDesc,
     bool isFatal)
```

Called when the underlying audio engine sends an error message to the SDK. Will contain a brief error name, a full description as well as a Boolean indicating whether or not the error is considered a fatal error.

```
void inputMeterChanged(uint8_t meter)
```

Called when the local SDK detects a change in the audio input level coming from the local user's audio source. The meter parameter will contain an integer between 0-100.

```
virtual void outputMeterChanged(uint8_t meter)
```

Called when the local SDK detects a change in the audio output level being sent to the local user's audio output device. The meter parameter will contain an integer between 0-100.

```
void participantAmplitudeChanged(const Participant participant,
     uint8_t amplitude, bool muted)
```

Called when the SDK detects that the volume level of a specific remote user has changed. The `amplitude` parameter will contain an integer between 0-100. If the `muted` parameter is "true", it means that the audio engine has detected that this user has muted themselves.

```
void logMessage(unsigned int level, const std::string& message)
```

If you've called `MeetingSDK::enableLogForwarding(true)`, any log messages received will be sent to your application through this callback. The `level` parameter contains the log level (1-7) and the `message` parameter contains the actual log message.

```
void binaryPlaybackEnded(uint64_t id)
```

Called when a sound played through the `MeetingSDK::playSound()` API call has finished playing.

```
void binaryPlaybackFailed(uint64_t id)
```

Called when a sound played through the `MeetingSDK::playSound()` API call has failed to either start or finish playing.

```
virtual void meetingDisconnected()
```

Called in the rare circumstance where the SDK has detected that communication with the back-end server has not happened within 60 seconds and, as such, the SDK has terminated the meeting connection.

```
void participantNetworkQuality(const Participant participant,
     std::string streamId, uint32_t barValue)
```

Called to update the application on the status of a given participant's network quality for a particular video stream. The `barValue` parameter is an integer from 0-5 with the following meanings:

0. No network detected
1. Major packet loss detected
2. A fair amount of packet loss detected
3. Some packet loss detected

4. Little packet loss detected
5. Negligible packet loss detected

`virtual void networkQuality(uint32_t barValue)`

Called to update the application on the status of the local user's network quality. The `barValue` parameter is an integer from 0-4. The integer value has the following meanings:

0. No network detected
1. Major packet loss detected
2. A fair amount of packet loss detected
3. Some packet loss detected
4. Little packet loss detected
5. Negligible packet loss detected

---

void connectionStatus(uint32_t status)

Called to update the application on the status of the current meeting connection. The integer received has the following meaning:

0. Connected to the meeting
1. In the process of connecting to the meeting
2. Disconnected from the meeting
3. Connection has failed
4. In the process of re-connecting to the meeting

MODERATOR SDK DELEGATE

The `ModeratorSDKDelegate` class represents an "interface" of member functions that may be called by the SDK to notify the application of certain events. It is defined as a concrete class with default implementations for each method that do nothing. To receive callbacks in your application, you must derive a new class from the `ModeratorSDKDelegate` class and implement the member functions you would like the SDK to call for you. Then call the `ModeratorSDK::setDelgate()` member function passing a pointer to the instance of your class. Each member function is listed below:

ModeratorSDKDelegate Member Functions:

```
void deviceListUpdated(const std::string& deviceJSON)
```

Called when the SDK receives an update (from the Visionable RTN service) regarding the devices of all users in the meeting. The list of users and devices is represented by a single JSON string passed as the sole parameter to this call.