Cordova Documentation

The EnableX Cordova Plugin is for developing Hybrid Mobile Applications (Cordova/PhoneGap/Ionic) integrated with EnableX RTC sessions. The Plugin has all the methods interfacing EnableX Signaling Server and Media Servers to communicate and to listen to event-based notifications during a session.

How to use EnableX Cordova Plugin?

By Creating New Application

1. Create New Application By Command
   cordova create SampleApp com.example.sampleapp videoconferencing
2. Add Platform
   cordova platform add ios cordova platform add android
3. Add EnableX Plugin into your Project
   cordova plugin add enablex-cordova-plugin
   To remove enablex plugin use following command.
   cordova plugin remove enablex-cordova-plugin
4. Build application
   cordova build
5. Run the Application
   cordova run ios
   Cordova run android

By Existing Enablex Sample Application.

1. Clone or download Repository Demo Application
2. In your terminal, change your directory to the root of the sample project that you want to run.
3. Run the command to install required node modules:
   npm install
4. Run the command to add Platforms
   cordova platform add iOS //to add iOS your project
   cordova platform add android // to add android project
5. Before running the application, you need to configure following variables
   var userName = "USERNAME"; /* HTTP Basic Auth Username of App Server */
   var password = "PASSWORD"; /* HTTP Basic Auth Password of App Server */
   var kBaseURL = "FQDN" /* FQDN of App Server URL */
6. Run the command to enable multidex.(Its only for android platform)
   cordova plugin add cordova-plugin-enable-multidex
7. Run the application by using following command
   cordova run ios // to run iOS project

cordova run android // to run Android project

Methods & Event Listeners:

Methods: Following are the method

1. JoinRoom(token,publishStreamInfo,roomInfo)
2. initLocalView(localviewOptions, successCallback, errorCallback)
3. initRemoteView(remoteviewOptions, successCallback, errorCallback)
4. muteSelfAudio(audio)
5. muteSelfVideo(video)
6. switchCamera(successCallback, errorCallback)
7. disconnect()
8. addScreenShare(viewoptions, successCallback, errorCallback)
9. removeScreenShare(successCallback, errorCallback)
10. addCanvasScreen(viewoptions, successCallback, errorCallback)
11. removeCanvasScreen(successCallback, errorCallback)
12. startDragging(view, drag, successCallback, errorCallback)
13. hideSelfView(hide, successCallback, errorCallback)
14. hideRemoteView(hide, successCallback, errorCallback)
15. hideScreenShareView(hide, successCallback, errorCallback)
16. hideCanvasScreen(hide, successCallback, errorCallback)
17. resizeLocalView(resizeOptions, successCallback, errorCallback)
18. resizeRemoteView(resizeOptions, successCallback, errorCallback)
19. switchMediaDevice(device)
20. getSelectedDevice(successCallback, errorCallback)
21. getDevices(successCallback, errorCallback)
22. getTalkerCount()
23. getMaxTalkers()
24. setTalkerCount(count)
25. postClientLogs()
26. enableLogs(enable)
27. muteSubscribeStreamsAudio(mute, successCallback, errorCallback)
28. setAudioOnlyMode(audioOnly, successCallback, errorCallback)
29. getReceiveVideoQuality(streamType, successCallback, errorCallback)
30. setReceiveVideoQuality(videoQualityOptions, successCallback, errorCallback)
31. adjustLayout(successCallback, errorCallback)
32. updateConfiguration(configuartionOptions, successCallback, errorCallback)
33. requestFloor()
34. cancelFloor()
35. finishFloor()
36. grantFloor(clientId)

37. denyFloor(clientId)
38. releaseFloor(clientId)
39. extendConferenceDuration()
40. lockRoom()
41. unLockRoom()
42. dropUser(jsonArray)
43. destroy()
44. hardMute()
45. hardUnMute()
46. startRecord()
47. stopRecord()
48. stopVideoTracksOnApplicationBackground(videoMuteRemoteStream, videoMuteLocalStream, successCallback, errorCallback)
49. startVideoTracksOnApplicationForeground(restoreVideoRemoteStream, restoreVideoLocalStream, successCallback, errorCallback)
50. enableStats(enableStats)
51. switchUserRole(clientId)
52. makeOutboundCall(text)
53. sendMessage(text, broadcast, array)
54. sendUserData(text, broadcast, array)
55. setAdvancedOptions(array)
56. getAdvancedOptions()
57. sendFiles(broadcast, array)
58. downloadFile(json, isAutoSave)
59. cancelUpload(jobId)
60. cancelDownload(jobId)
61. cancelAllUploads()
62. cancelAllDownloads()
63. getAvailableFiles(successCallback, errorCallback)
64. getClientId(successCallback, errorCallback)
65. getRoomId(successCallback, errorCallback)
66. getClientName(successCallback, errorCallback)
67. getLocalStreamID(successCallback, errorCallback)
68. getUserList(successCallback, errorCallback)
69. getRoomMetadata(successCallback, errorCallback)
70. isConnected(successCallback, errorCallback)
71. enableProximitySensor(status)
72. getMode(successCallback, errorCallback)
73. getRole(successCallback, errorCallback)
74. whoAmI(successCallback, errorCallback)
75. startAnnotation(clientid)
76. stopAnnotations()

Event Listeners:

Following is the method to add event listeners

1. addEventListner(EventName, Callback)

Event Name -

1. onRoomConnected
2. onRoomError
3. onUserConnected
4. onUserDisConnected
5. onEventError
6. onEventInfo
7. onAcknowledgedSendData
8. onMessageReceived
9. onSwitchedUserRole
10. onUserRoleChanged
11. onConferencessExtended
12. onConferenceRemainingDuration
13. onAckDropUser
14. onAckDestroy
15. onAudioEvent
16. onVideoEvent
17. onRoomDisConnected
18. onNotifyDeviceUpdate
19. onBandWidthUpdated
20. onShareStreamEvent
21. onCanvasStreamEvent
22. onAckLockRoom
23. onAckUnLockRoom
24. onLockedRoom
25. onUnLockedRoom
26. onLogUploaded
27. onHardMutedAudio
28. onHardUnMutedAudio
29. onReceivedHardMuteAudio
30. onReceivedHardUnMuteAudio
31. onHardMutedVideo
32. onHardUnMutedVideo
33. onReceivedHardMuteVideo
34. onReceivedHardUnMuteVideo
35. onHardMuted
36. onReceivedHardMute

37. onHardUnMuted
38. onReceivedHardUnMute
39. onConnectionInterrupted
40. onConnectionLost
41. onOutBoundCallInitiated
42. onDialStateEvents
43. onPlayerStats
44. onReconnect
45. onUserReconnectSuccess
46. onStartRecordingEvent
47. onRoomRecordingOn
48. onStopRecordingEvent
49. onRoomRecordingOff
50. onAcknowledgeStats
51. onReceivedStats
52. onSetTalkerCount
53. onGetTalkerCount
54. onMaxTalkerCount
55. onRemoteStreamAudioMute
56. onRemoteStreamAudioUnMute
57. onRemoteStreamVideoMute
58. onRemoteStreamVideoUnMute
59. OnCapturedView
60. onAdvancedOptionsUpdate
61. onGetAdvancedOptions
62. onFloorRequested
63. onFloorRequestReceived
64. onProcessFloorRequested
65. onGrantedFloorRequest
66. onDeniedFloorRequest
67. onReleasedFloorRequest
68. onFileUploadStarted
69. onFileAvailable
70. onInitFileUpload
71. onFileUploaded
72. onFileUploadCancelled
73. onFileUploadFailed
74. OnFileDownloaded
75. onFileDownloadCancelled
76. onFileDownloadFailed
77. onInitFileDownload
78. onUserDataReceived

79. onCanvasStarted
80. onCanvasStopped
81. onScreenSharedStarted
82. onScreenSharedStopped
83. onAnnotationStarted
84. onStartAnnotationAck
85. onAnnotationStopped
86. onStoppedAnnotationAck
87. OnStartCanvasAck
88. onStoppedCanvasAck

Methods & Event Listeners Descriptions:

Join a Room with Stream -

To quick start and join the room use join room method. JoinRoom method comes as a handy tool for developer. It handles all complexities of connecting and joining the room.

Method - JoinRoom(token,publishStreamInfo,roomInfo,success,error)

```
@param {String} token -it is encoded token string received from Enx
application server.
@param {JSON} publishStreamInfo - publishStreamInfo for local streams
@param {JSON} - roomInfo - room info for joining room
@param {SuccessCallback} -
@param {ErrorCallback} -
```

EventListener –

- `onRoomConnected` – When Client End Point is connected to the room successfully
- `onRoomDisConnected` – Client End Point got disconnected to the room
- `onRoomError` – Client End Point's attempt to connect to room has failed
- `onUserConnected` – Everyone is notified that a new user is connected to the Room
- `onUserDisConnected` – Everyone is notified that a connected user is disconnected from the Room
- `onConnectionLost` – When End Point looses network connection
- `onConnectionInterrupted` – When connection is interrupted e.g Switch from WiFi to 4G and vice versa
- `onUserReconnectSuccess` – When End-Point successfully gets reconnected with EnableX

- `onReconnect` – When End-Point trying to reconnect within given time period.
- `onPublishedStream` – Publisher is notified that its Stream has been published into the Room
- `onUnPublishedStream` – Publisher is notified that its Stream is unpublished/removed from the Room.

Code Snippets.

```
var videoSize = {
  minWidth: 320,
  minHeight: 180,
  maxWidth: 1280,
  maxHeight: 720,
};

var streamOpt = {
  audio: true,
  video: true,
  data: true,
  audioOnlyMode: false,
  framerate: 30,
  maxVideoBW: 1500,
  minVideoBW: 150,
  videoSize: videoSize,
  audioMuted: false,
  videoMuted: false,
  maxVideoLayers: 1,
  name: "Shashank"
};

var playerConfiguration = {
  audiomute: true,
  videomute: true,
  bandwidth: true,
  screenshot: true,
  avatar: true,
  iconHeight: 30,
  iconWidth: 30,
```

```
          avatarHeight: 200,
          avatarWidth: 200,
        };

      var roomOpt = {
          activeviews: "list",
          allow_reconnect: true,
          number_of_attempts: 3,
          timeout_interval: 15,
          playerConfiguration: playerConfiguration,
          forceTurn: false,
          chat_only: false,
        };

     window.EnxRtc.joinRoom(result.token, streamOpt,roomOpt, function
(data) {
          console.log( JSON.stringify(data.data));
          initLocalView(); // To init local View
          initRemoteView(); // To init Remote View
        }, function (err) {
            }
);
```

```
// Add Event Listeners
window.EnxRtc.addEventListner("onRoomConnected", function      (data) {
          console.log( JSON.stringify(data.data));
        });
window.EnxRtc.addEventListner("onRoomError", function      (data) {
          console.log( JSON.stringify(data.data));
          initLocalView(); // To init local View
          initRemoteView(); // To init Remote View
        });

window.EnxRtc.addEventListner("onEventError", function (data) {
          console.log(JSON.stringify(data.data));
        });
```

```
window.EnxRtc.addEventListner("onUserConnected", function (data) {
          console.log(JSON.stringify(data.data));
        });

window.EnxRtc.addEventListner("onUserDisConnected", function (data) {
          console.log( JSON.stringify(data.data));
        });
```

Init Local View and Remote View

These methods are used to show local stream and remote streams both. Plugin automatically manages local view and remote view. Users must init local view and remote view in the success of room connected event.

Method:

1. `initLocalView(localviewOptions, successCallback, errorCallback)`
2. `initRemoteView(remoteviewOptions, successCallback, errorCallback)`

Code Snippets

```
   // To Init Local View
   var initLocalViewOptions = {
        height: 130,
        width: 100,
        margin_top: 50,
        margin_left: 0,
        margin_right: 15,
        margin_bottom: 10,
        position: "top"
      };

 window.EnxRtc.initLocalView(initLocalViewOptions,    function
(data) {
       console.log(JSON.stringify(data.data));
      }, function (err) {
        console.log('Uh oh... error' + JSON.stringify(err));
```

```
      });


   // To init Remote View

   var initRemoteViewOptions = {
        height: 600,
        width: 800,
        margin_top: 100,
        margin_left: 0,
        margin_right: 15,
        margin_bottom: 10,
        position: "center"

      };
  window.EnxRtc.initRemoteView(initRemoteViewOptions, function
(data) {
        console.log( JSON.stringify(data.data));
      }, function (err) {
        console.log(JSON.stringify(err));
      });
```

Mute / Unmute Audio

Use `window.EnxRtc.muteSelfAudio()` method to mute and unmute audio from Local Stream. When a user mutes or unmutes audio from own Published Stream, Self user notified with event `onAudioEvent` and all other connected users of the room are notified with event listeners `onRemoteStreamAudioMute` and `onRemoteStreamAudioUnMute` callbacks respectively. Listen to these events to update related UI elements.

Methods:

`muteSelfAudio( audio)` –

@param {Boolean} audio - Pass true to mute, false to unmute audio

EventListeners:

- `onRemoteStreamAudioMute` – to all participants notifying user has muted audio
- `onRemoteStreamAudioUnMute` – to all participants notifying user has unmuted audio
- `onAudioEvent` – to self that audio is either muted or unmuted

Code Snippets:

```
  // To mute audio
  window.EnxRtc.muteSelfAudio(true);

  // To unmute audio
  window.EnxRtc.muteSelfAudio(false);

  // Add event listeners

  // To self. Audio is muted/unmuted.
  window.EnxRtc.addEventListner("onAudioEvent", function (data)    {
    console.log(JSON.stringify(data.data));
   });


   // To all. Audio muted by Remote user
  window.EnxRtc.addEventListner("onRemoteStreamAudioMute", function (data)
{
    console.log(JSON.stringify(data.data));
   });



   // To all. Audio unmuted by Remote user
  window.EnxRtc.addEventListner("onRemoteStreamAudioUnMute", function
(data)    {
    console.log(JSON.stringify(data.data));
```

```
  });
```

Mute / Unmute Video in a Stream

Use `window.EnxRtc.muteSelfVideo()` method to mute and unmute video from Local Stream. When a user mutes or unmutes video from own Published Stream, self user notified with event `onVideoEvent` and all all other connected users of the room are notified with `onRemoteStreamVideoMute` and `onRemoteStreamVideoUnMute` callbacks respectively.

Methods:

`muteSelfVideo( video)`

@param {Boolean} video- Pass true to mute, false to unmute video

Callbacks:

- `onRemoteStreamVideoMute` – to all participants notifying user has muted video
- `onRemoteStreamVideoUnMute` – to all participants notifying user has unmuted video
- `onVideoEvent` – to self that video is either muted or unmuted

Code Snippets:

```
// To mute video
window.EnxRtc.muteSelfVideo(true);

// To unmute video
window.EnxRtc.muteSelfVideo(false);
   //Add event listeners
```

```
  // To self. Video is muted/unmuted.
  window.EnxRtc.addEventListner("onVideoEvent", function (data)     {
    console.log(JSON.stringify(data.data));
   });

  // To all. Video muted by Remote user
  window.EnxRtc.addEventListner("onRemoteStreamVideoMute", function (data)
{
    console.log(JSON.stringify(data.data));
   });

  // To all. Video unmuted by Remote user
  window.EnxRtc.addEventListner("onRemoteStreamVideoUnMute", function
(data)     {
    console.log(JSON.stringify(data.data));
   });
```

Switch between Rear & Front Camera

If user looks to switch between Rear and Front Camera as a source for published Stream, use `window.EnxRtc.switchCamera()` method.

Method:

`switchCamera()` – No parameter needed

Code Snippets:

```
    window.EnxRtc.switchCamera(false, function (data) {

          console.log(JSON.stringify(data.data));

          }, function (err) {
```

```
            console.log(JSON.stringify(err));

          });
```

Disconnect from a Room

A Client End Point can disconnect itself from the room to close its session. A disconnected End-Point will loose media and signaling socket immediately.

Once disconnected, the Toolkit will receive callback `onRoomDisconnected` for you to handle UI. Also, all connected users of the session will receive callback `onUserDisconnected` so that they all can update the status of their UI.

Method:

`disconnect()` – Without any parameter

EventListener:

- `onRoomDisconnected` – To the user who is disconnected
- `onUserDisconnected` – To all other connected users

Code Snippets:

```
window.EnxRtc.disconnect();

// Notified User who is disconnected.
window.EnxRtc.addEventListner("onRoomDisConnected",  function (data)    {
   console.log(JSON.stringify(data.data));
  });

 // Other users in the room is notified.
```

```
  window.EnxRtc.addEventListner("onUserDisConnected",  function (data)
{
    console.log(JSON.stringify(data.data));
   });
```

Screen Share EventListeners

EventListeners:

- `onScreenSharedStarted` – All participants are notified that Screen Share has started
- `onScreenSharedStopped` – All participants are notified that Screen Share has stopped

Methods:

- `addScreenShare(viewOptions, successCallback, errorCallback)` - this method is used to show a screen share in the room. To show the share screen call this function in the event listener `onScreenSharedStarted`.
- `removeScreenShare(successCallback, errorCallback)` - this method is used to remove screen shared in the room. TO remove Screen Shared call this function in the event listener `onScreenSharedStopped`.

Code Snippets

```
    // Add event Listener to listen screen shared started in the room
```

```
window.EnxRtc.addEventListner("onScreenSharedStarted",  function (data)
{
    console.log(JSON.stringify(data.data));
     addScreenShare(); // Add Screen Share
   });

  // Add event Listener to listen screen shared stopped in the room
window.EnxRtc.addEventListner("onScreenSharedStopped",  function (data)
{
    console.log(JSON.stringify(data.data));
     removeScreenShare(); // Add Screen Share
   });



 // To Add Screen Share



    var options = {
        height: 130,
        width: 100,
        margin_top: 50,
        margin_left: 0,
        margin_right: 15,
        margin_bottom: 10,
        position: "top"
      };

      window.EnxRtc.addScreenShare(options, function (data) {
        console.log('Excelsior succuss! ' + JSON.stringify(data.data));

      }, function (err) {
        console.log('Uh oh... error' + JSON.stringify(err));
      });



      // To Remove Screen Share

       window.EnxRtc.removeScreenShare(function (data) {
```

```
    console.log('Excelsior succuss! ' + JSON.stringify(data.data));

  }, function (err) {
    console.log('Uh oh... error' + JSON.stringify(err));
  });
```

Canvas EventListeners

EventListeners:

- `onCanvasStarted`– All participants are notified that Canvas has started
- `onCanvasStopped` – All participants are notified that Canvas has stopped

Methods:

- `addCanvasScreen(viewOptions, successCallback, errorCallback)` - this method is used to show a canvas in the room. To show the canvas, call this function in the event listener `onCanvasStarted`.
- `removeCanvasScreen(successCallback, errorCallback)` - this method is used to remove screen shared in the room. TO remove Screen Shared call this function in the event listener `onCanvasStopped`.

Code Snippets

```
```

```
      // Add event Listener to listen canvas started in the room
window.EnxRtc.addEventListner("onCanvasStarted",  function (data)    {
    console.log(JSON.stringify(data.data));
           addCanvasScreen(); // Add Canvas
   });

  // Add event Listener to listen canvas stopped in the room
window.EnxRtc.addEventListner("onCanvasStopped",  function (data)    {
    console.log(JSON.stringify(data.data));
     removeCanvasScreen(); // Add Canvas
   });



 // To Add Canvas



    var options = {
        height: 130,
        width: 100,
        margin_top: 50,
        margin_left: 0,
        margin_right: 15,
        margin_bottom: 10,
        position: "top"
      };

      window.EnxRtc.addCanvasScreen(options, function (data) {
        console.log('Excelsior succuss! ' + JSON.stringify(data.data));

      }, function (err) {
        console.log('Uh oh... error' + JSON.stringify(err));
      });



      // To Remove Canvas

       window.EnxRtc.removeCanvasScreen(function (data) {
        console.log('Excelsior succuss! ' + JSON.stringify(data.data));
```

```
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Hide/ UnHide local view

Use `window.EnxRtc.hideSelfView()` method to hide and unhide local view in the running conference.

Methods:

```
hideSelfView(hide, successCallback, errorCallback)

@param {boolean} hide - Pass true for hide and false for unhide
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
// To hide local view
window.EnxRtc.hideSelfView(true,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });

// To unhide local view
window.EnxRtc.hideSelfView(false,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Hide/ UnHide Remote view

Use `window.EnxRtc.hideRemoteView()` method to hide and unhide remote view in the running conference.

Methods:

```
hideRemoteView(hide, successCallback, errorCallback)

@param {boolean} hide - Pass true for hide and false for unhide
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
// To hide Remote view
window.EnxRtc.hideRemoteView(true,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });

// To unhide Remote view
window.EnxRtc.hideRemoteView(false,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Hide/ UnHide ScreenShare view

Use `window.EnxRtc.hideScreenShareView()` method to hide and unhide screen share view in the running conference.

Methods:

```
hideScreenShareView(hide, successCallback, errorCallback)

@param {boolean} hide - Pass true for hide and false for unhide
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
// To hide screen share view
window.EnxRtc.hideScreenShareView(true,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });

// To unhide screen share view
window.EnxRtc.hideScreenShareView(false,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Hide/ UnHide Canvas view

Use `window.EnxRtc.hideCanvasScreen()` method to hide and unhide canvas view in the running conference.

Methods:

```
hideCanvasScreen(hide, successCallback, errorCallback)

@param {boolean} hide - Pass true for hide and false for unhide
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
// To hide Canvas view
window.EnxRtc.hideCanvasScreen(true,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });

// To unhide Canvas view
window.EnxRtc.hideCanvasScreen(false,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Resize local view

Use `window.EnxRtc.resizeLocalView()` method to resize the local view in the running conference.

Methods:

```
resizeLocalView(resizeOptions, successCallback, errorCallback)

@param {JSON} resizeOptions
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
var options = {
      height: 130,
      width: 100,
      margin_top: 50,
      margin_left: 0,
      margin_right: 15,
      margin_bottom: 10,
      position: "top"
    };
window.EnxRtc.resizeLocalView(options ,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Resize Remote views

Use `window.EnxRtc.resizeRemoteView()` method to resize the remote view in the running conference.

Methods:

```
resizeRemoteView(resizeOptions, successCallback, errorCallback)

@param {JSON} resizeOptions
```

```
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
var options = {
      height: 130,
      width: 100,
      margin_top: 50,
      margin_left: 0,
      margin_right: 15,
      margin_bottom: 10,
      position: "center"
    };
window.EnxRtc.resizeRemoteView(options ,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Switch to alternate audio devices

To Switch Media Device

If user looks to switch Micrcophone, Bluetooth devices or speaker. Use `window.EnxRtc.switchMediaDevice()` method. This API requires ID of the new device to switch to. You can use `window.EnxRtc.getDevices()` method to fetch a list of all audio devices and use one of these audio device name to switch audio device..

Method:

`switchMediaDevice(Device)`

`@param {String} device -` It should be device name getting from getDevice api.

EventListener

`onNotifyDeviceUpdate` – When Audio Device changes is complete. Returns with switched Microphone Device name

Code Snippet:

```
    // deviceName is getting from getDevice api

 window.EnxRtc.switchMediaDevice(deviceName, function (data) {

      console.log(JSON.stringify(data.data));

}, function (err) {

   console.log('Uh oh... error resizeLocalView ' + JSON.stringify(err));

          });



      // To listen event listener

      window.EnxRtc.addEventListner("onNotifyDeviceUpdate",  function
(data)    {

    console.log(JSON.stringify(data.data));
   });
```

To Get All Audio Device

If user looks to get all connected audio device like earpiece, Bluetooth devices or speaker. Use `window.EnxRtc.getDevices()` method. This API returns array of device.

Method:

```
getDevices(successCallback, errorCallback)

@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

```
window.EnxRtc.getDevices(function (data) {
          console.log(JSON.stringify(data.data));
        }, function (err) {
          console.log('JSON.stringify(err));
        });
```

To Get Current Selected device

If user looks to get current selected audio device. Use `window.EnxRtc.getSelectedDevice()` method. This API returns device name.

Method:

```
getSelectedDevice(successCallback, errorCallback)

@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getSelectedDevice(function (data) {
          console.log(JSON.stringify(data.data));
        }, function (err) {
          console.log('JSON.stringify(err));
        });
```

Get maximum permissible Talker Count

To know maximum permissible Active Talkers that you may receive and you can set, you may use `window.EnxRtc.getMaxTalkers()` method.

Method:

`getMaxTalkers()` – without parameter

EventListener

`onMaxTalkerCount`

Code Snippet

```
window.EnxRtc.getMaxTalkers();

window.EnxRtc.addEventListner("onMaxTalkerCount",    function (data)     {
  console.log(JSON.stringify(data.data));
 });
```

Get Talker Count

It may be necessary to know how many talkers are expected to receive i.e. to know either the preset value of talkers or any custom value in effect which is either by the Application Developer or opted by End-Point User.

Method:

`getTalkerCount()` – without parameter

EventListener

`onGetTalkerCount`

Code Snippets

```
window.EnxRtc.getTalkerCount();
window.EnxRtc.addEventListner("onGetTalkerCount",   function (data)    {
  console.log(JSON.stringify(data.data));
 });
```

Set Talker Count

EnableX sets 6 active talkers by default in the active talker-list. However, you may opt to receive less talkers at a Client End Point if you so require. This may either be a predefined value, or this value may be set at run-time in a connected session. If needed, you can create UI for the connected user to opt for number of active-talkers the user wants to receive.

Method:

`setTalkerCount(numTalkers)`

`@param {Number} numTalkers -` No. of talkers you want to receive. Range 0-6

- If you set `numTalkers` to any value from 1 to 6 – you will receive those many talkers in the list.
- If you set `numTalkers` to 0 (zero), then list doesn't become empty. Rather, you would start receiving 3 audio streams only. Video Streams will not be carried.

EventListener

`onSetTalkerCount`

Code Snippets

```
```

```
window.EnxRtc.setTalkerCount(4);
window.EnxRtc.addEventListner("onSetTalkerCount",   function (data)    {
  console.log(JSON.stringify(data.data));
 });
```

Share Log with EnableX to audit

To share Console Log with EnableX Tech Team for auditing purpose, use `window.EnxRtc.postClientLogs()` . The method sends latest 200KB of Console Log to EnableX.

Before you can share Console Log, you need to enable Console logging by `window.EnxRtc.enableLogs(value)`

Method:

`enableLogs()` – To enable logs of Enablex SDK.

`@param {boolean} enable` Pass true to enable and false to disable.

`postClientLogs()` – to share log. No parameter needed

EventListener:

`onLogUploaded`

Code Snippets:

```
// To enable Logs
window.EnxRtc.enableLogs(true);

// To post logs
window.EnxRtc.postClientLogs();

// Add Listener to listen success
window.EnxRtc.addEventListner("onLogUploaded",   function (data)    {
  console.log(JSON.stringify(data.data));
```

```
  });
```

Mute Subscribe Streams Audio:

To Mute All remote stream audio on the client side. Use `window.EnxRtc.muteSubscribeStreamsAudio()`

Method:

```
muteSubscribeStreamsAudio(mute, successCallback, errorCallback)
@param {boolean} mute - pass true for mute and false for unmute.
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
window.EnxRtc.muteSubscribeStreamsAudio(mute ,function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Change to Audio only call

If you want change to a audio-only call, i.e. you neither want to receive anyone's video not want to publish your video; you may use `window.EnxRtc.setAudioOnlyMode()` method. This method acts as a toggle to switch between audio-only and audio-video call.

This opt-in is only for the specific End-Point and doesn't affect others in the room.

Method:

`setAudioOnlyMode(audioOnly,successCallback, errorCallback)`

`@param {boolean} audioOnly - pass true for audio-only call and false for audio-video call.`
`@param {CallableFunction} successCallback`
`@param {CallableFunction} errorCallback`

```
window.EnxRtc.setAudioOnlyMode(audioOnly ,function (data) {
      console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
      console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Opt to receive desired Video Quality

The Client End Point may opt to receive the desired video quality for available bandwidth. You may create UI based on the enumerated values of video quality as explained below.

Method:

`setReceiveVideoQuality(videoQualityOptions, successCallback, errorCallback) - To set desired video quality.`

`@param {JSON} videoQualityOptions`
`@param {CallableFunction} successCallback`
`@param {CallableFunction} errorCallback`
`videoQualityOptions` – JSON Object may have the following keys:

- `videoQuality`: Enumerated Values: `Auto`, `HD`, `SD`, `LD`. Set it to `Auto` if you want EnableX to decide optimum quality for you dynamically.
- `streamType`: Enumerated Values: `talker`, `canvas`

Code Snippets**`streamType`**

```
 var videoQualityOptions = {
       streamType: "Auto",
       videoQuality: "talker"
     };


window.EnxRtc.setReceiveVideoQuality(videoQualityOptions ,function (data)
{
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

`getReceiveVideoQuality(streamType, successCallback, errorCallback)`- To get current selected video quality.

`@param {String} streamType` - It should be talker or canvas

`@param {CallableFunction} successCallback`

`@param {CallableFunction} errorCallback`

Code Snippets:

```
window.EnxRtc.getReceiveVideoQuality("talker" ,function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Adjust Layout

This Method helps to adjust Video Player Layout in case you have received complete Active Talker View. In case user rotates Screen, this method will help Video layout to re-adjust to fit within its Parent View.

Method:

```
adjustLayout(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
window.EnxRtc.adjustLayout(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Update Configuration

If you like to re-configure your stream by adding new or udpating existing specification parameters of a stream. Please use `window.EnxRtc.updateConfiguration()` method to update the parameters of a stream. Both remote and local streams can use this method of updating the stream parameters..

Method:

```
updateConfiguration(configuarationOptions,successCallback,
errorCallback)
```

```
@param {JSON} configuartionOptions
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:**maxAudioBW**

```
 var videoQualityOptions = {
       maxVideoBW: “900”,
       minVideoBW: “150”,
       maxAudioBW: “150”,
       minAudioBW: “150”
      };

window.EnxRtc.updateConfiguration(configuarationOptions,function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Floor Access Control in Lecture Mode

In Lecture Mode, only the Moderator publishes a stream to the room whereas all other participants subscribe to see and listen to the Moderator. If participants have questions to ask during the session, the Moderator can grant the floor to the participant to publish his stream.

The following steps to deploy Control Floor Access by moderator:

Request Floor Access

A participant requests for Floor access is forwarded to the Moderator using a notification event.

Method:

`requestFloor()` – No parameter needed. Participant requests floor access

EventListeners:

- `onFloorRequested` – Requesting Participant received acknowledgement that his request has been received by Moderator
- `onFloorRequestReceived` – The Moderator receives requests from Participant

Code Snippets:

```
    // To request floor access

     window.EnxRtc.requestFloor();

    // Add Event Listeners



    // Your Request is received by Moderator.
    window.EnxRtc.addEventListner("onFloorRequested",   function   (data)
{
    console.log(JSON.stringify(data.data));
   });

   // Moderator receives floor access request
   window.EnxRtc.addEventListner("onFloorRequestReceived",   function
(data)    {
    console.log(JSON.stringify(data.data));
   });
```

Cancel Requested Floor Access

A participant may cancel his request for Floor access which is pending at Moderator Side.

Method:

`cancelFloor()` - Participant cancels request floor access

EventListeners:

- `onCancelledFloorRequest:` Moderator receives cancellation notification for already posted Floor Access Request from Participant
- `onFloorCancelled:` Participant receives acknowledgement that his request for Floor Request has been cancelled.

Code Snippets:

```
    // TO cancel request floor
    window.EnxRtc.cancelFloor();

    // Add Event Listeners

// Participant is acknowledged that floor request is cancelled.
    window.EnxRtc.addEventListner("onFloorCancelled",   function   (data)
{
    console.log(JSON.stringify(data.data));
   });

   // Moderator receives cancellation floor request.
   window.EnxRtc.addEventListner("onCancelledFloorRequest",   function
(data)    {
    console.log(JSON.stringify(data.data));
   });
```

Grant Floor Access

Moderator might grant floor access to one or many users one by one. Note that at any given point of time, only one person may be granted floor access. To grant access to others, moderator must release floor access from the participant who is currently granted floor access. Once granted floor access, the designated participant will be notified using an event and he can publish his stream.

Method:

`grantFloor(clientID)` – moderator grants floor access to participant

`@param {String} clientID -` It's the Client ID for the participant whom access is being granted.

EventListeners:

- `onProcessFloorRequested` – Moderator is acknowledged that Participant is granted floor access
- `onGrantedFloorRequest` – Participant receives floor access

Code Snippets:

```
    // To grant floor access to a client id
    window.EnxRtc.grantFloor(clientID);

    // Add Event Listeners



    // Moderator granted floor access to a participant.
    window.EnxRtc.addEventListner("onProcessFloorRequested",   function
(data)    {
    console.log(JSON.stringify(data.data));
   });

   // Participant is notified that he is granted floor access.
   window.EnxRtc.addEventListner("onGrantedFloorRequest",   function
(data)    {
    console.log(JSON.stringify(data.data));
```

```
   });
```

Finish Granted Floor Access

The Participant can finish floor access granted to him by Moderator.

Method:

`finishFloor()` – Participant finishes his granted Floor Access

EventListeners:

- `onFinishedFloorRequest:` Moderator receives notification that participant has finish granted Floor Access himself.
- `onFloorFinished:` Participant receives acknowledgement that he finished granted Floor Access.

Code Snippets:

```
    // To grant floor access to a client id
    window.EnxRtc.finishFloor();

    // Add Event Listeners

   // Participant receive acknowledgement that he finished granted floor
access.
   window.EnxRtc.addEventListner("onFloorFinished",   function (data)    {
    console.log(JSON.stringify(data.data));
   });


    // Moderator notified that granted floor access to a participant has
finished.
```

```
    window.EnxRtc.addEventListner("onFinishedFloorRequest",    function
(data)     {
    console.log(JSON.stringify(data.data));
   });
```

Deny Floor Access

The Moderator may also deny a request for floor access to a participant.

Method:

`denyFloor(String ClientID)` – Moderator denies floor access to a Participant

`@param {String} clientID -` It's the Client ID for the participant

EventListeners:

- `onProcessFloorRequested` – Moderator is acknowledged that he denied floor access to a Participant
- `onDeniedFloorRequest` – Participant is notified that he has been denied floor access

Code Snippets:

```
    // To deny floor access to a client id
    window.EnxRtc.denyFloor(clientID);

    // Add Event Listeners



    // Moderator notified when he denied floor access.
    window.EnxRtc.addEventListner("onProcessFloorRequested",    function
(data)     {
    console.log(JSON.stringify(data.data));
   });

   // Participant is notified that moderator is granted floor access.
   window.EnxRtc.addEventListner("onDeniedFloorRequest",    function (data)
{
```

```
    console.log(JSON.stringify(data.data));
   });
```

Release Floor Access

Moderator may terminate floor access granted to some participant. Upon termination, the user's stream(s) is unpublished. It is necessary to terminate floor access of a participant before giving out floor access to another participant.

Method:

`releaseFloor(clientId)` – Moderator asks participant to release floor

`@param {String} clientID -` It's the Client ID for the participant

EventListeners:

- `onProcessFloorRequested` – Moderator is acknowledged that floor is released
- `onReleasedFloorRequest` – Participant i notified that floor access has been revoked. His stream is automatically unpublished.

Code Snippets:

```
    // To release floor access to a client id
    window.EnxRtc.releaseFloor(clientID);

    // Add Event Listeners



    // Moderator notified when he released floor access.
    window.EnxRtc.addEventListner("onProcessFloorRequested",    function
(data)     {
    console.log(JSON.stringify(data.data));
   });
```

```
  // Participant is notified that moderator is released floor access.
   window.EnxRtc.addEventListner("onReleasedFloorRequest",   function
(data)    {
    console.log(JSON.stringify(data.data));
   });
```

Extend Conference Duration

Lock / Unlock Room

Moderator may lock the Room forbidding any new person from joining the Session. To allow subsequent users to join session, Moderator need to unlock the Room.

To put room into Lock State, you use `window.EnxRtc.lockRoom()` method; and to unlock the Room you use `window.EnxRtc.unLockRoom()` method.

Methods:

- `lockRoom()` – To lock Room
- `unLockRoom()` – To unlock Room

EventListeners:

- `onAckLockRoom:` To acknowledge moderator that the Room has been locked
- `onLockedRoom:` To notify all participants that the room has been locked
- `onAckUnLockRoom:` To acknowledge moderator that the Room has been unlocked

- `onUnLockedRoom:` To notify all participants that the room has been unlocked

Code Snippets:

```
    // To lock Room
    window.EnxRtc.lockRoom();

    // Add Event Listeners

    // Moderator is acknowledged that room has been locked.
    window.EnxRtc.addEventListner("onAckLockRoom", function  (data)    {
    console.log(JSON.stringify(data.data));
   });

   // Participants are notified that room has been locked.
   window.EnxRtc.addEventListner("onLockedRoom",   function (data)    {
    console.log(JSON.stringify(data.data));
   });

     // TO unlock Room

   window.EnxRtc.unLockRoom();

    // Add Event Listeners

    // Moderator is acknowledged that room has been unlocked.
    window.EnxRtc.addEventListner("onAckUnLockRoom",   function  (data)
{
    console.log(JSON.stringify(data.data));
   });



  // Participants are notified that room has been unlocked.
   window.EnxRtc.addEventListner("onUnLockedRoom",   function (data)    {
    console.log(JSON.stringify(data.data));
   });
```

Disconnect User

Moderation may wish to disconnect or force-drop one or more participant(s) from the Session by using `window.EnxRtc.dropUser()` method.

Method:

`dropUser(clientIdsArray)`

`@param {String} clientID -` It's array of CientId to be disconnected.

EventListeners:

- `onAckDropUser:` To acknowledge that the user has been disconnected
- `onRoomDisConnected:` To notify the effected user that he is disconnected from Room with reason.

Code Snippets:

```
// To disconnect the user from room
Var array = [c1,c2];
window.EnxRtc.dropUser(array);


// Add Event Listeners

// Moderator is acknowledged that user has been dropped.
window.EnxRtc.addEventListner("onAckDropUser", function  (data)    {
console.log(JSON.stringify(data.data));
});
```

```
   // Participants are notified that he is disconnected from the room with
reason.
   window.EnxRtc.addEventListner("onRoomDisConnected",   function (data)
{
    console.log(JSON.stringify(data.data));
   });
```

Destroy Session

Moderation may wish to conclude an ongoing session by using using `window.EnxRtc.destroy()` method.

Method:

`destroy():` No Parameter.

EventListener:

- `onAckDestroy` To acknowledge that the session has been destroyed
- `onRoomDisconnected:` To notify all users that Session is closed.

Code Snippets:

```
 // To destroy Room

   window.EnxRtc.destroy();


   // Add Event Listeners
```

```
    // Moderator is acknowledged that Session has been destroyed.
    window.EnxRtc.addEventListner("onAckDestroy", function  (data)    {
    console.log(JSON.stringify(data.data));
   });

   // Participants are notified that session is closed.
   window.EnxRtc.addEventListner("onRoomDisConnected",   function (data)
{
    console.log(JSON.stringify(data.data));
   });
```

On-Demand Recording

Client API call helps you start and stop recording as and when needed. The available methods are accessible to Moderator only. Therefore, on the Moderator End-Point, you may require to create UI to use the methods to start or stop recording a session.

To start session recording on demand you may use `window.EnxRtc.startRecord()` method, and to stop may use `window.EnxRtc.stopRecord()` method. On successful initiation of recording and stopping of recording the moderator will be notified using events `onStartRecordingEvent` and `onStopRecordingEvent`. also, all other participants will be notified using events `onRoomRecordingOn` and `onRoomRecordingOff` respectively.

Methods:

- `startRecord()` – No parameter needed. To start recording
- `stopRecord()` – No parameter needed. To stop recording

EventListeners:

- `onStartRecordingEvent` – To moderator to notify that recording has started
- `onStopRecordingEvent` – To moderator to notify that recording has stopped
- `onRoomRecordingOn` – To all participant to notify recording is on
- `onRoomRecordingOff` – To all participant to notify recording is off

Code Snippets:

```
  // To Start Recording

    window.EnxRtc.startRecord();


    // Add Event Listeners

    // Moderator is acknowledged that Recording has been started.
    window.EnxRtc.addEventListner("onStartRecordingEvent", function
(data)     {
    console.log(JSON.stringify(data.data));
   });

   // All Participants are notified that Recording has been started.
   window.EnxRtc.addEventListner("onRoomRecordingOn
",    function (data)     {
    console.log(JSON.stringify(data.data));
   });


      // To Stop Recording

    window.EnxRtc.stopRecord();


    // Add Event Listeners

    // Moderator is acknowledged that Recording has been stopped.
```

```
    window.EnxRtc.addEventListner("onStopRecordingEvent", function  (data)
{
    console.log(JSON.stringify(data.data));
   });

   // All Participants are notified that Recording has been stopped.
   window.EnxRtc.addEventListner("onRoomRecordingOff",   function (data)
{
    console.log(JSON.stringify(data.data));
   });
```

Note:

- Moderator can start/stop recording any number of time during a session as he wishes.
- Moderator can stop recording in a room defined with auto-recording feature.

Hard Mute / Unmute Room

Moderator may put the room in a hard-mute state when he wishes no-one else to be audible in the Room. Any new user who joins the room when the room is in hard-mute state will also be inaudible

To put room into hard mute state, you may use `window.EnxRtc.muteRoom()` method; and to disable you may use `window.EnxRtc.unMuteRoom()` method. All participant of the room are notified with events `onReceivedMuteRoom` and `onReceivedUnMuteRoom` whereas the Moderator is notified with events `onMutedRoom` and `onUnMutedRoom` respectively.

Methods:

- `muteRoom()` – to hard mute room
- `unMuteRoom()` – to hard unmute room

EventListeners:

- `onReceivedMuteRoom` – To all participants on hard-muting Room
- `onReceivedUnMuteRoom` – To all participants on hard-unmuting Room
- `onMutedRoom` – To notify Moderator that Room is hard-muted
- `onUnMutedRoom` – To notify Moderator that Room is hard-unmuted

Code Snippets:

```
    // To hard-mute Room
    window.EnxRtc.hardMute();



  // Add Event Listeners

    // Moderator is acknowledged that Room has been muted.
    window.EnxRtc.addEventListner("onHardMuted", function  (data)    {
    console.log(JSON.stringify(data.data));
   });

   // All Participants are notified that Room has been muted.
   window.EnxRtc.addEventListner("onReceivedHardMute",   function (data)
{
    console.log(JSON.stringify(data.data));
   });


      // To hard-unmute Room

    window.EnxRtc.hardUnMute();


    // Add Event Listeners

    // Moderator is acknowledged that Room has been unmuted.
    window.EnxRtc.addEventListner("onHardUnMuted", function  (data)    {
    console.log(JSON.stringify(data.data));
```

```
   });

   // All Participants are notified that Room has been unmuted.
   window.EnxRtc.addEventListner("onReceivedHardUnMute",   function (data)
{
    console.log(JSON.stringify(data.data));
   });
```

Handle Application Switch from Foreground to Background

User may switch to different application pushing your RTC Application to Background and vice versa. You need to handle such activity using following methods:

Method:

```
stopVideoTracksOnApplicationBackground(localMuteState,remoteMuteStat
e ,successCallback, errorCallback)
```

`@param {boolean} localMuteState` – Boolean. Pass *true* to pause local Video Stream, *false* to continue with publishing
`@param {boolean} remoteMuteState` – Boolean. Pass *true* to pause receiving remote Video Stream, *false* to continue receiving
`@param {CallableFunction} successCallback`
`@param {CallableFunction} errorCallback`

Method:

```
startVideoTracksOnApplicationForeground(localUnMuteState,remoteUnMut
eState ,successCallback, errorCallback )
```

`@param {boolean} localUnMuteState` – Boolean. Pass true to resume sending local Video Stream if it was stopped while getting to Background.
`@param {boolean} remoteUnmuteState` – Boolean. Pass true to resume receiving remote Video Streams if they stopped while getting to Background.
`@param {CallableFunction} successCallback`

```
@param {CallableFunction} errorCallback
```

Code Snippets:

```
    // To pause

window.EnxRtc.stopVideoTracksOnApplicationBackground(true,true,function
(data) {
        console.log('Excelsior succuss! ' + JSON.stringify(data.data));
      }, function (err) {
        console.log('Uh oh... error' + JSON.stringify(err));
      });

      // To Resume

window.EnxRtc.startVideoTracksOnApplicationForeground
(true,true,,function (data) {
        console.log('Excelsior succuss! ' + JSON.stringify(data.data));
      }, function (err) {
        console.log('Uh oh... error' + JSON.stringify(err));
      });
```

Receive Streaming Stats

For Streaming Quality analysis, you may opt to receive Streaming Stats and get them displayed on respective Player. Once opted, you will receive Stats for all Streams

received at the end point from the Room. You will receive following important information on Streams:

- Publishing Resolution
- Bandwidth at Publisher End
- Receiving Resolution
- Receiving Bandwidth consumed
- Available Bandwidth at Receiver
- … there are more

Method:

`enableStats(isEnabled)` – To enable disable streaming stats for all streams in the Room.

`@param {boolean} isEnabled` – To enable disable streaming stats for all streams in the Room.

EventListener:

- `onAcknowledgeStats` – Acknowledges that stream stats is enabled or disabled
- `onReceivedStats` – When all streams' stats is received
- onPlayerStats - When Player's stream stats is received

Code Snippets:

```
    // To enable stats
    window.EnxRtc.enableStats(true);


      // Add Event Listeners

    // Acknowledged that stream stats is enable or disable.
    window.EnxRtc.addEventListner("onAcknowledgeStats", function  (data)
{
```

```
 console.log(JSON.stringify(data.data));
});


// All Streams stats.
window.EnxRtc.addEventListner("onReceivedStats",function (data)    {
 console.log(JSON.stringify(data.data));
});

// Particular Streams stats.
window.EnxRtc.addEventListner("onPlayerStats",function (data)    {
 console.log(JSON.stringify(data.data));
});
```

Change Participant's Role

A Moderator can designate a connected Participant to act as a Moderator during the session. Doing so, the promoted Participant would get access to all the rights of the Moderator, i.e. accesses all features listed under Moderator Controls; and the old Moderator becomes a Participant. Similarly, the new Moderator can further grant Moderator Role to other participant if needed.

Method:

`switchUserRole(ClientId)`

`@param {boolean} isEnabled` – Client ID of the participant who is being designated as Moderator

EventListner:

- `onSwitchedUserRole` – Moderator is acknowledged that he has requested for Switch User Role
- `onUserRoleChanged` – All participants are notified with a message that a new user has been appointed as moderator. Also, the newly appointed moderator is notified with extra information

Code Snippets:

```
    // To enable stats
    window.EnxRtc.switchUserRole(clientId);


      // Add Event Listeners

    // Moderator is acknowledged.
    window.EnxRtc.addEventListner("onSwitchedUserRole", function  (data)
{
    console.log(JSON.stringify(data.data));
   });


   // Client whose role is changed from participant to moderator.
   window.EnxRtc.addEventListner("onUserRoleChanged",function (data)    {
    console.log(JSON.stringify(data.data));
   });
```

Make Outbound Call

This Api is used for outbound call like

Method:

`makeOutboundCall(text)`

`@param {String} text`– text will be mobile number with std code.

EventListner:

- `onOutBoundCallInitiated`– user is acknowledged that he called.
- `onDialStateEvents`– Participants are notified that user called on that number.

Code Snippets:

```
    // To enable stats
    window.EnxRtc.makeOutboundCall(clientId);

      // Add Event Listeners

    window.EnxRtc.addEventListner("onOutBoundCallInitiated", function
(data)    {
    console.log(JSON.stringify(data.data));
   });

   window.EnxRtc.addEventListner("onDialStateEvents",function (data)    {
    console.log(JSON.stringify(data.data));
   });
```

Private, Public & Group Messaging

This is an advance messaging feature between Session participants. You can do following type of messaging in a session among participants:

- **Public Messaging:** To send message to all connected users.
- **Private Messaging:** To send message to a specific user.
- **Group Messaging:** To send message to more than one specified users.

Messaging feature neither require the sender to publish his Stream, nor receivers to subscribe.

Method:

`sendMessage(message, isBroadcast, recipientIDs)`

`@param {String} text -` String type message.
`@param {boolean} broadcast -` Boolean. Use *true* for Public Broadcast, Use *false* for private messaging to one or more recipients.
`@param {Array} RecipientIDs` – Array of ClientIDs whom you wish to send private messages.
.
EventListeners:

- `onAcknowledgedSendData`– acknowledges message is sent as a JSONObject.
- `onMessageReceived` – Receives message in JSONObject.

Code Snippets:

```
// To send message
message = "hi";
var broadcast = true; // Incase of public messaging
Var broadcast = false; // Incase of private messaging.
var array = []; // Incase of public messaging
```

```
    Var array = [clientiid_1]; // Incase of private messaging.
    window.EnxRtc.sendMessage( message,true,array);

      // Add Event Listeners

    window.EnxRtc.addEventListner("onAcknowledgedSendData", function
(data)     {
    console.log(JSON.stringify(data.data));
   });

   window.EnxRtc.addEventListner("onMessageReceived",function (data)     {
    console.log(JSON.stringify(data.data));
   });
```

Use Custom Signaling

Your Application might require to send instructions, data to one or more recipient connected in a Session to deploy new features, business workflow. For example, you want to create a Polling mechanism among participants. EnableX supports Custom Signaling method through which you can build such utility that requires passing of messages among participants.

Using Custom Signaling Method, you might send message to all or selected participants in a Room. You can define your custom data structure to pass to meet your business requirement.

Method:

`sendUserData( data, isBroadcast,RecipientIDs)`

`@param {String} data-` JSONObject containing custom keys. This object is passed to Recipients as is. EnableX doesn't enforce it's structure. Be advised to define keys effectively for signaling needs.

```
var data = {
```

```
    "sender": "Sender's Name",
    "message": "Message body",
    "custom_key": "Data"
}
```

`@param {boolean} broadcast -` Boolean. Use *true* for Public Broadcast, Use *false* for private messaging to one or more recipients.
`@param {Array} RecipientIDs` – Array of ClientIDs whom you wish to send private messages.

EventListener:

- `onReceivedChatDataRoom` – Receives signaling in JSONObject. Available until Android Toolkit v1.5.2. Deprecated on v1.5.3
- `onUserDataReceived` – Receives signaling in JSONObject. Available from Android Toolkit v1.5.3

Code Snippets:

```
    // To send message
    message = “”; // JSON
    var broadcast = true; // Incase of public messaging
    Var broadcast = false; // Incase of private messaging.
    var array = []; // Incase of public messaging
    Var array = [clientiid_1]; // Incase of private messaging.
    window.EnxRtc.sendUserData( message,true,array);

      // Add Event Listeners


  window.EnxRtc.addEventListner("onAcknowledgedSendData", function  (data)
{
    console.log(JSON.stringify(data.data));
   });

   window.EnxRtc.addEventListner("onUserDataReceived",function (data)     {
    console.log(JSON.stringify(data.data));
   });
```

Set Advance Options

To set advanced options like battery updated,video quality changed etc.

Method:

```
setAdvancedOptions( array )
```

```
@param {Array} array - Its a array of advanced options
```

EventListener:

- `onAdvancedOptionsUpdate` – Receives when advance options is setted and called any update on video quality or battery changed.

Code Snippets

:

```
      window.EnxRtc.setAdvancedOptions ( array);

      // Add Event Listeners

  window.EnxRtc.addEventListner("onAdvancedOptionsUpdate", function
(data)     {
    console.log(JSON.stringify(data.data));
   });
```

Get Advance Options

To get advanced options use `window.EnxRtc.getAdvancedOptions()`

Method:

`getAdvancedOptions()-` No Parameter is required.

EventListener:

- `onGetAdvancedOptions` – Receives when you called getAdvancedOptions is called.

Code Snippets:

```
      window.EnxRtc.getAdvancedOptions ( );

      // Add Event Listeners

  window.EnxRtc.addEventListner("onGetAdvancedOptions", function  (data)
{
    console.log(JSON.stringify(data.data));
   });
```

File Sharing

EnableX File Share API allows users of a RTC session to send and receive file(s) to/from each other. Using the available APIs, you can initiate file transfer to share, cancel a file transfer, get notified on a shared file and receive/download shared file.

Upload File to share

The file sharing process starts with a user initiating a file transfer to EnableX Server. To initiate a file transfer use `window.EnxRtc.sendFiles()` method.

Method:

```
sendFiles( broadcast, arra)
```

```
@param {boolean} broadcast -broadCast This is to share file among all participants in the Session. If you need to target the file share to specific user, then set it to false.

@param {Array} array - array of Client IDs. This is to share the file among specified Clients only. If broadcast is set to true, clientIdList is ignored.
```

EventListener at Sender End:

- `onInitFileUpload` – To notify sender that file upload process is initiated
- `onFileUploaded` – To notify sender that file has been uploaded
- `onFileUploadFailed` – To notify sender that file upload process has failed

EventListener at Receiver End:

- `onFileUploadStarted` – To notify intended receiver that a file is being uploaded
- `onFileAvailable` – To notify intended received that a file is ready to download

Code Snippets:

```
      window.EnxRtc. sendFiles ( broadcast, array);

      // Add Event Listeners

      // To intended receiver - A new file upload started
  window.EnxRtc.addEventListner("onFileUploadStarted", function  (data)
{
    console.log(JSON.stringify(data.data));
   });

   // To intended receiver - A file is avilable for download
window.EnxRtc.addEventListner("onFileAvailable", function  (data)    {
    console.log(JSON.stringify(data.data));
   });

      // To intended receiver - File upload process has initialized.
window.EnxRtc.addEventListner("onInitFileUpload", function  (data)    {
    console.log(JSON.stringify(data.data));
   });


      // To intended receiver - File upload is complete.
window.EnxRtc.addEventListner("onFileUploaded", function  (data)    {
    console.log(JSON.stringify(data.data));
   });


      // To intended receiver - File upload has failed.
window.EnxRtc.addEventListner("onFileUploadFailed", function  (data)    {
    console.log(JSON.stringify(data.data));
   });
```

Error Codes: Upload process may encounter error. You will be notified with JSON Object with error code:

Error Code	Error Description
5089	Storage Access denied
5091	File Sharing not available in this context
1185	Too large file. Max allowed Size: NNNN
1182	Failed to upload file

Download shared File

Intended Recipients needs to download each file shared with them individually. Therefore, recipient needs to know information on the available files to download to initiate a download process.

Know files to download

Room Lookup: To know what all files are available for download, you may call `window.EnxRtc.getAvailableFiles()` method. It returns a JSON Object with all files available for download

Method:

```
getAvailableFiles(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippets:

```
window.EnxRtc.getAvailableFiles(function (data) {
        console.log('Excelsior succuss! ' + JSON.stringify(data.data));
      }, function (err) {
        console.log('Uh oh... error' + JSON.stringify(err));
      });
```

When New File is availble: A receive end point is notified as and when a file is made available for download using success callback.

Initiate File Download

As you know about the file(s) to download, you need to initiate download individually using `window.EnxRtc.downloadFile()` method.

Method:

```
downloadFile(fileInfo, isAutoSave)
@param {JSON} json - JSON Object of file to be downloaded.
@param {boolean} isAutoSave - Whether to save the file automatically. If not to be save automatically, you would receive Base64 encoded RAW data to handle file saving processes manually. If auto-saving is on, you would receive saved file path.
```

EventListener at Receiver End:

- `onInitFileDownload` - To notify that file download process is initiated
- `onFileDownloaded` – To notify file has been downloaded with either Base64 Raw data to be saved (When auto-saving is false) or saved file path.
- `onFileDownloadFailed` – To notify download failure

Code Snippets:

```
    window.EnxRtc.downloadFile(JSON, isAutoSave);

    // Add Event Listeners

    // To intended receiver - File download process has initialized.
window.EnxRtc.addEventListner("onInitFileDownload", function (data) {
  console.log(JSON.stringify(data.data));
 });
```

```
   // To receiver - File download has failed
window.EnxRtc.addEventListner("onFileDownloadFailed", function  (data)
{
    console.log(JSON.stringify(data.data));
   });

      // To intended receiver - File has been downloaded.
window.EnxRtc.addEventListner("onFileDownloaded", function  (data)    {
    console.log(JSON.stringify(data.data));
   });
```

Error Codes: Download process may encounter error. You will be notified with JSON Object with error code:

Error Code	Error Description
5089	Storage Access denied
5090	Failed to save file
1183	Failed to download file
1181	File Download not available in this context

Cancel file Share

User cancel thier uploading file by this API.

Method:

`cancelUpload(jobId)` - To cancel single uploading file
`@param {Number} jobId`

`CancelAllUploads()` - To cancel all uploading files.
No params required.

EventListener

- `onFileUploadCancelled` – To notify file has been cancelled.

Code Snippets

```
```

```
 // To cancel single uploading file
 window.EnxRtc.cancelUpload(jobId);

// To cancel all uploading file
 window.EnxRtc.cancelAllUploads();


      // Add Event Listeners

      // To intended receiver - File uploading is cancelled.
  window.EnxRtc.addEventListner("onFileUploadCancelled", function  (data)
{
    console.log(JSON.stringify(data.data));
   });
```

Cancel file Share

User cancel thier uploading file by this API.

Method:

`cancelDownload(jobId)` - To cancel single downloading file
`@param {Number} jobId`

`cancelAllDownloads()` - To cancel all downloading files.
No params required.

EventListener

- `onFileDownloadCancelled` - To notify file has been cancelled.

Code Snippets

```
 // To cancel single downloading file
 window.EnxRtc.cancelDownload(jobId);

// To cancel all downloading file
```

```
window.EnxRtc.cancelAllDownloads();


     // Add Event Listeners

     // To intended receiver - File downloading is cancelled.
  window.EnxRtc.addEventListner("onFileDownloadCancelled", function  (data)
{
    console.log(JSON.stringify(data.data));
   });
```

Get connected User ID or Client ID

Each connected user to the Room is assigned with a Unique Client ID for the session. To know the Client ID of the connected user from the enc-point, use `window.EnxRtc.getClientId()` method.

Method:

```
getClientId(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getClientId (function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Get Room ID

Every Room is assigned with a Unique Room ID while creating the Room. The Room ID of the room to which you are connected may be known by using `window.EnxRtc.getRoomId()` method.

Method:

```
getRoomId(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getRoomId(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Get connected User Name

To know the name of connected user from the end-point, use `window.EnxRtc.getClientName()` method.

Method:

```
getClientName(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getClientName(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
```

```
    });
```

Get Local Stream ID

Every Stream is identified by its Stream ID, be it a local or a remote stream, a canvas stream or a screen share stream. You may obtain the ID of the Local Stream using `window.EnxRtc.getLocalStreamID()` method.

Method:

```
getLocalStreamID(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getLocalStreamID(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
    }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
    });
```

Get connected User List

To get a list of connected users to the Room to which the enc-point is also connected to, use `window.EnxRtc.getUserList()` method. It returns a JSON with list of connected user's information.

Method:

```
getUserList(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getUserList(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Get Room Meta Information

To know Meta Information of the Room to which you are connected, you may use window.EnxRtc.getRoomMetaData() method. Room Meta information contains complete Room Definition along with many Run-Time parameters with their current values & room stats of connected users and streams. This is received by all end-points after getting connected to Room along with onRoomConnected eventlistener. Some of the run-time parameters are updated internally by the Toolkit on various events.

Method:

```
getRoomMetaData(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getRoomMetadata(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Check Room is connected or not

To know status of the Room that is connected or not, you may use window.EnxRtc.isConnected() method.

Method:

```
isConnected(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.isConnected(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Get Room Mode

A connected Room may be on *group* or *lecture* mode. Use `window.EnxRtc.getMode()` to know mode of Room.

Method:

```
getMode(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getMode(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Get connected User Role

A User connected to a Room with either role of *moderator* or *participant*. Use `window.EnxRtc.getRole()` to know role of the user.

Method:

```
getRole(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.getRole(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Get connected User Information at an End-Point

To get connected user information at an End-Point, use `window.EnxRtc.whoami()` method. It returns complete user-meta for the connected user in a JSON Object.

Method:

```
whoami(successCallback, errorCallback)
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Code Snippet

```
window.EnxRtc.whoAmI(function (data) {
       console.log('Excelsior succuss! ' + JSON.stringify(data.data));
     }, function (err) {
       console.log('Uh oh... error' + JSON.stringify(err));
     });
```

Enable Proximity Sensor

EnxProximitySensor manages functions related to the "Proximity Sensor". On most device, the Proximity Sensor is implemented as a boolean-sensor.

- It returns just two values NEAR or FAR.
- Thresholding is done on the LUX value i.e. the LUX value of the light Sensor is compared with a threshold.
- A LUX-value more than the threshold means the Proximity Sensor returns FAR.
- Anything less than the threshold value, the Sensor returns NEAR

Method:

```
enableProximitySensor (value ,successCallback, errorCallback)

@param {boolean} status - true to enable and false to disable
Proximity Sensor.
@param {CallableFunction} successCallback
@param {CallableFunction} errorCallback
```

Start / Stop Annotation

To initiate Annotation, you need to add eventListener after getting connected to the room.

Methods:

- startAnnotation(clientId) – To start Annotation on given clientId

  ```
  @param {String} clientid
  ```

- stopAnnotation() – To stop Annotation

EventListeners:

- onStartAnnotationAck(JSONObject) – Acknolwedgement to the Annotator that Annotation has started
- onAnnotationStarted(EnxStream) – Notification to all others that Annotation has started
- onStoppedAnnotationAck(JSONObject) – Acknolwedgement to the Annotator that Annotation has stopped
- onAnnotationStopped(EnxStream) – Notification to all others that Annotation has stopped