Sportstalk Android SDK

Lawrence C. Cendana

Nov 10, 2020

CONTENTS:

CHAPTER

ONE

GETTING STARTED

Use this API to create experiences powered by SportsTalk and interact with those experiences.

- All API calls should be made using HTTPS.
- The API is designed to minimize the number of requests you need to make so chat applications are able to serve users very quickly especially for mobile users who need the lowest possible latency

1.1 Implementing the SDK

You can download the latest SportsTalk Android SDK from the following location:

https://gitlab.com/sportstalk247/sdk-android-kotlin

You need to register SportsTalk API with 'Appkey' and 'Token'.

1.2 How to get API Key and Token

You need to visit the dashboard with the following URL:

https://dashboard.sportstalk247.com

Then click on ''Application Management" link to generate the above

1.3 Authentication

- All requests that require authentication must have the x-api-token header with your application token.
- Most requests require authentication.
- If you provide authentication on a request that does not require authentication the header will be ignored and it will have no effect.

1.4 SDK flow for Chat Applications

The typical flow of an application is:

- *Create Room*: Creates a chat room.
- *List Rooms*: Returns a list of available rooms. If you know the ID of the room you want there is no need to invoke List All Rooms first.
- *Join Room*: A user joins the room as an anonymous user or as a logged in user. Only logged in users can engage in chat activities. Anonymous users can only view whats happening in the room. You can also use custom room IDs that you provide so that you can use naming conventions to and join rooms without needing to call List All Rooms to get a room ID first.
- *Get Updates*: This gets the most recent events that have occurred in a room. You can use this endpoint as often as you want for polling, or you can use the Firebase API to get more bandwidth efficent push events when updates occur in the room. Calling Get Updates will prevent a logged in user from being removed from the room due to inactivity.
- *Execute Chat Command*: This performs a command in a chat room, like when you run a program from the command line. By default, the command is to say something in the room. However if a command starts with a special character such as / you can perform an action. See the Execute Chat Command API for more details and possible commands.
- *Exit Room*: When a logged in user exits the room, call this event. Otherwise the user will be removed from the room after some time without any activity.
- *List Participants*: Lists the logged in users in the chat room.
- *Get Room Details*: Gets statistics about the room such as the number of participants.

CHAPTER

TWO

HOW TO DOWNLOAD THE SDK

2.1 Download from Repository

The SportsTalk SDK has been published into **jitpack.io**. In order to use it in your application, just do the following:

1. Add the following in root **build.gradle** file

```
allprojects {
    repositories {
    // ...
       maven {
          url "https://jitpack.io"
       }
    }
}
```

1. Add the following lines in your module **build.gradle** file, under dependencies section

```
implementation 'com.gitlab.sportstalk247:sdk-android-kotlin:vX.Y.Z'
```

Release

Then sync again. The gradle build should now be successful.

CHAPTER

THREE

HOW TO USE SPORTSTALK SDK

3.1 Instantiate SportsTalkManager Clients

This Sportstalk SDK is meant to power custom chat applications. Sportstalk does not enforce any restricitons on your UI design, but instead empowers your developers to focus on the user experience without worrying about the underlying chat behavior.

```
class MyFragment: Fragment() {

    // ...
    // ...

    // YOUR APP ID
    val appId = "c84cb9c852932a6b0411e75e" // This is just a sample app id
    // YOUR API TOKEN
    val apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA" // This is just a␣
↪sample token
    val endpoint = "http://api.custom.endpoint/v1/" // please ensure out of the box␣
↪the SDKs are configured for production URL

    // Instantiate User Client
    val userClient = SportsTalk247.UserClient(
        config = ClientConfig(
            appId = appId,
            apiToken = apiToken,
            endpoint = endpoint
        )
    )

    // Instantiate Chat Client
    val chatClient = SportsTalk247.ChatClient(
        config = ClientConfig(
            appId = appId,
            apiToken = apiToken,
            endpoint = endpoint
        )
    )

    // ...

}
```

3.2 Use SDK with Coroutines

Android Sportstalk SDK is an Asynchronous-driven API, powered by Kotlin Coroutines) to gracefully handle asynchronous operations.

Client SDK functions are declared with `suspend` keyword. This means that the function should be invoked from within coroutine scope. See the example below:

```
class MyFragment: Fragment() {

    // ...
    // ...
    // Instantiate User Client
    val userClient = SportsTalk247.UserClient(/*...*/)

    // Instantiate Chat Client
    val chatClient = SportsTalk247.ChatClient(/*...*/)

    override fun onViewCreated(view: View) {
        // ...
        // Launch thru coroutine block
        // https://developer.android.com/topic/libraries/architecture/coroutines
        lifecycleScope.launch {
            // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
            val createdUser = withContext(Dispatchers.IO) {
                userClient.createOrUpdateUser(
                    request = CreateUpdateUserRequest(
                                userid = "8cb689cc-21b7-11eb-adc1-0242ac120002", //␣
↪sample user ID
                                handle = "sample_handle_123",
                                displayname = "Test Name 123", // OPTIONAL
                                pictureurl = "https://i.imgur.com/oh1x5wW.jpeg", //␣
↪OPTIONAL
                                profileurl = "https://i.imgur.com/oh1x5wW.jpeg" //␣
↪OPTIONAL
                            )
                )
            }

            // Resolve `createdUser` from HERE onwards(ex. update UI displaying the␣
↪response data)...
        }

    }

}
```

3.3 Handling SDK Exception

If any client operations receive an error response, whether it be Network, Server, or Validation Error, these functions will throw an instance of `SportsTalkException`.

```
data class SportsTalkException(
   val kind: String? = null, // "api.result"
   val message: String? = null, // ex. "The specified comment was not found."
   val code: Int? = null // ex. 404,
   val data: Map<String, String?>? = null,
   val err: Throwable? = null
)

// Under Fragment class
// Execute within coroutine scope
lifecycleScope.launch {
   val testComment = Comment(id = "0987654321",...)

   val setCommentDeletedResponse = try {
      withContext(Dispatchers.IO) {
         // These should throw Error 404 - "The specified conversation was not found
↪and was not deleted.".
         commentClient.permanentlyDeleteComment(
            conversationid = "Non-existent-Conversation-ID",
            commentid = testComment.id!!
         )
      }
   } catch(err: SportsTalkException) {
      // Resolve ERROR from HERE.
      return
   }
}
```

CHAPTER

FOUR

USER CLIENT

4.1 Create or Update User

Invoke this function if you want to create a user or update an existing user.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#8cc680a6-6ce8-4af7-ab1e-e793a7f0e7d2

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val createdUser = withContext(Dispatchers.IO) {
        userClient.createOrUpdateUser(
        request = CreateUpdateUserRequest(
                    userid = "023976080242ac120002",
                    handle = "sample_handle_123",
                    displayname = "Test Name 123", // OPTIONAL
                    pictureurl = "<Image URL>", // OPTIONAL
                    profileurl = "<Image URL>" // OPTIONAL
                )
        )
    }

    // Resolve `createdUser` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

4.2 Get User Details

This will return all the information about the user.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#3323caa9-cc3d-4569-826c-69070ca51758

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val userDetails = withContext(Dispatchers.IO) {
        userClient.getUserDetails(
            userid = "023976080242ac120002"
        )
    }

    // Resolve `userDetails` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

4.3 List Users

Use this function to cursor through a list of users. This function will return users in the order in which they were created, so it is safe to add new users while cursoring through the list.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#51718594-63ac-4c28-b249-8f47c3cb02b1

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
```

(continues on next page)

(continued from previous page)

```
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val listUsers = withContext(Dispatchers.IO) {
        userClient.listUsers(
            limit = 10, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of user(s).
        )
    }

    // Resolve `listUsers` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

4.4 Ban User

This function toggles the specified user's `banned` flag.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#211d5614-b251-4815-bf76-d8f6f66f97ab

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val bannedUser = withContext(Dispatchers.IO) {
        userClient.setBanStatus(
            userid = "023976080242ac120002",
            banned = true // If set to true, attempt to ban the user. If set to false,
↪ attempt to remove the ban from user
        )
    }

    // Resolve `bannedUser` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

4.5 Shadow Ban User

This function toggles the specified user's `shadowbanned` flag.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#211a5696-59ce-4988-82c9-7c614cab3efb

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val shadowBannedUser = withContext(Dispatchers.IO) {
        userClient.shadowBanUser(
            userid = "023976080242ac120002",
            shadowban = true, // If set to true, user can send messages into a chat␣
↪room, however those messages are flagged as shadow banned.
            expireseconds = 3600 // [OPTIONAL]: Duration of shadowban value in␣
↪seconds. If specified, the shadow ban will be lifted when this time is reached. If␣
↪not specified, shadowban remains until explicitly lifted. Maximum seconds is a␣
↪double byte value.

        )
    }

    // Resolve `shadowBannedUser` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

4.6 Search User(s)

This function searches the users in an app.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#dea07871-86bb-4c12-bef3-d7290d762a06

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
```

(continues on next page)

(continued from previous page)

```
    )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    // Search by Handle
    val searchedUsersByHandle = withContext(Dispatchers.IO) {
        userClient.searchUsers(
            handle = "testhandle1",
            limit = 20, // Defaults to 200 on backend API server
            cursor = null   // OPTIONAL: The cursor value from previous search␣
↪attempt to indicate next paginated fetch. Null if fetching the first list of␣
↪user(s).
        )
    }

    // Search by Name
    val searchedUsersByName = withContext(Dispatchers.IO) {
        userClient.searchUsers(
            name = "Josie Rizal",
            limit = 20, // Defaults to 200 on backend API server
            cursor = null   // OPTIONAL: The cursor value from previous search␣
↪attempt to indicate next paginated fetch. Null if fetching the first list of␣
↪user(s).
        )
    }

    // Search by User ID
    val searchedUsersByUserId = withContext(Dispatchers.IO) {
        userClient.searchUsers(
            userid = "userid_georgew",
            limit = 20, // Defaults to 200 on backend API server
            cursor = null   // OPTIONAL: The cursor value from previous search␣
↪attempt to indicate next paginated fetch. Null if fetching the first list of␣
↪user(s).
        )
    }
}
```

4.7 Delete User

This function will delete the specified user. All rooms with messages by that user will have the messages from this user purged in the rooms.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#ab387784-ad82-4025-bb3b-56659129279c

Below is a code sample on how to use this SDK feature:

```
val userClient = SportsTalk247.UserClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
```

(continues on next page)

(continued from previous page)

```
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val deletedUser = withContext(Dispatchers.IO) {
        userClient.deleteUser(
            userid = "023976080242ac120002"
        )
    }

    // Resolve `deletedUser` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

CHAPTER

FIVE

CHAT CLIENT

5.1 Create Room

Invoke this function to create a new chat room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#8b2eea78-82bc-4cae-9cfa-175a00a9e15b

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val createdRoom = withContext(Dispatchers.IO) {
        chatClient.createRoom(
            request = CreateChatRoomRequest(
                name = "Test Chat Room 1",
                customid = "test-room-1",
                description = "This is a test chat room 1.",
                moderation = "post",
                enableactions = true,
                enableenterandexit = true,
                enableprofanityfilter = false,
                delaymessageseconds = 0L,
                roomisopen = true,
                maxreports = 0
            )
        )
    }

    // Resolve `createdRoom` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

5.2 Get Room Details

Invoke this function to get the details for a room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#9bac9724-7505-4e3e-966f-08cfebbca88d

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val chatRoom = withContext(Dispatchers.IO) {
        chatClient.getRoomDetails(
            chatRoomId = "080001297623242ac002"
        )
    }

    // Resolve `chatRoom` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

5.3 Get Room Details By CustomId

Invoke this function to get the details for a room, using custom ID.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#0fd07be5-f8d5-43d9-bf0f-8fb9829c172c

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
```

(continues on next page)

(continued from previous page)

```
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val chatRoom = withContext(Dispatchers.IO) {
        chatClient.getRoomDetailsByCustomId(
            chatRoomCustomId = "custom-id-0239760802"
        )
    }

    // Resolve `chatRoom` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

5.4 List Rooms

Invoke this function to list all the available public chat rooms.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#0580f06e-a58e-447a-aa1c-6071f3cfe1cf

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val listRooms = withContext(Dispatchers.IO) {
        chatClient.listRooms(
            limit = 20, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of chat room(s).
        )
    }

    // Resolve `chatRoom` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

5.5 Join Room (Authenticated User)

Invoke this function to join a room.

You want your chat experience to open fast. The steps to opening a chat experience are:

1. Create Room
2. Create User
3. Join Room (user gets permission to access events data from the room)
4. Get Recent Events to display in your app

If you have already created the room (step 1) then you can perform steps 2 - 4 using join room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#eb3f78c3-a8bb-4390-ab25-77ce7072ddda

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val joinRoomResponse = withContext(Dispatchers.IO) {
        chatClient.joinRoom(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = JoinChatRoomRequest(
                userid = "023976080242ac120002" // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `joinRoomResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.6 Join Room By CustomID

Invoke this function to join a room by Custom ID. This method is the same as Join Room, except you can use your customid.

The benefit of this method is you don't need to query to get the roomid using customid, and then make another call to join the room. This eliminates a request and enables you to bring your chat experience to your user faster.

You want your chat experience to open fast. The steps to opening a chat experience are:

1. Create Room
2. Create User
3. Join Room (user gets permission to access events data from the room)
4. Get Recent Events to display in your app

If you have already created the room (step 1) then you can perform steps 2 - 4 using join room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#a64f2c32-6167-4639-9c32-413edded2c18

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val joinRoomResponse = withContext(Dispatchers.IO) {
        chatClient.joinRoomByCustomId(
            chatRoomCustomId = "custom-room-id-12976",    // Custom ID of an existing␣
↪chat room
            request = JoinChatRoomRequest(
                userid = "023976080242ac120002" // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `joinRoomResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.7 Join Room (Anonymous User)

Invoke this function to join a room as an anonymous user.

A user can be added to a room in a logged in state or in an anonymous state. Typically the anonymous state is used so that people can see what is happening in the room and be enticed to register with you in order to participate in the conversation, as they must be logged in to say something or react to anything happening in the room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#c83c1afc-300b-4a18-b7e2-e3a1797dbca3

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val joinRoomResponse = withContext(Dispatchers.IO) {
        chatClient.joinRoom(
            chatRoomIdOrLabel = "080001297623242ac002"    // ID of an existing chat␣
↪room
        )
    }

    // Resolve `joinRoomResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.8 List Room Participants

Invoke this function to list all the participants in the specified room.

Use this method to cursor through the people who have subscribe to the room.

To cursor through the results if there are many participants, invoke this function many times. Each result will return a cursor value and you can pass that value to the next invokation to get the next page of results. The result set will also include a next field with the full URL to get the next page, so you can just keep reading that and requesting that URL until you reach the end. When you reach the end, no more results will be returned or the result set will be less than maxresults and the next field will be empty.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#1b1b82a9-2b2f-4785-993b-baed6e7eba7b

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val listRoomParticipants = withContext(Dispatchers.IO) {
        chatClient.listRoomParticipants(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            limit = 20, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of chatroom␣
↪participant(s).
        )
    }

    // Resolve `listRoomParticipants` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.9 Update Room

Invoke this function to update an existing room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#96ef3138-4820-459b-b400-e9f25d5ddb00

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val updatedRoom = withContext(Dispatchers.IO) {
        chatClient.updateRoom(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = UpdateChatRoomRequest(
```

(continues on next page)

(continued from previous page)

```
                name = "${testData.name!!}-updated",
                customid = "${testData.customid}-updated(${System.currentTimeMillis()}
↪)",
                description = "${testData.description}-updated",
                enableactions = !testData.enableactions!!,
                enableenterandexit = !testData.enableenterandexit!!,
                maxreports = 30L
            )
        )
    }

    // Resolve `updatedRoom` from HERE onwards(ex. update UI displaying the response␣
↪data)...
}
```

5.10 Execute Chat Command (say 'Hello, World!')

Invoke this function to execute a command in a chat room.

Precondition: The user must JOIN the room first with a call to Join Room. Otherwise you'll receive HTTP Status Code PreconditionFailed (412)

5.10.1 SENDING A MESSAGE

- Send any text that doesn't start with a reserved symbol to perform a SAY command.
- Use this function to REPLY to existing messages
- Use this function to perform ACTION commands
- Use this function to perform ADMIN commands

example:

```
ExecuteChatCommandRequest(
    command = "These commands both do the same thing, which is send the message
↪'Hello World' to the room. SAY Hello, World Hello, World",
    // ....
)
```

5.10.2 ACTION COMMANDS

- Action commands start with the / character

example:

```
// Assuming current user's handle is "@MikeHandle05"
ExecuteChatCommandRequest(
    command = "/dance nicole",
    // ....
)

// User sees: "You dance with Nicole"
```

(continues on next page)

(continued from previous page)

```
// Nicole sees: "@MikeHandle05 dances with you"
// Everyone else sees: "@MikeHandle05 dances with Nicole"
```

This requires that the action command dance is on the approved list of commands and Nicole is the handle of a participant in the room, and that actions are allowed in the room.

5.10.3 ADMIN COMMANDS

- These commands start with the * character

example:

```
// This bans the user from the entire chat experience (all rooms).
ExecuteChatCommandRequest(
    command = "*ban",
    // ....
)
```

```
// This restores the user to the chat experience (all rooms).
ExecuteChatCommandRequest(
    command = "*restore",
    // ....
)
```

```
// This deletes all messages from the specified user.
ExecuteChatCommandRequest(
    command = "*purge",
    // ....
)
```

```
// This deletes all messages in this room.
// Assuming ADMIN password "testpassword123"
ExecuteChatCommandRequest(
    command = "*deleteallevents testpassword123",
    // ....
)
```

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#c81e90fc-1a54-40bb-a75b-2fc935c12b59

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
```

(continues on next page)

(continued from previous page)

```
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val executeChatCmdResponse = withContext(Dispatchers.IO) {
        chatClient.executeChatCommand(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = ExecuteChatCommandRequest(
                command = "Hello World!",
                userid = "023976080242ac120002" // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `executeChatCmdResponse` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.11 Execute Chat Command (Announcement by Admin)

Invoke this function to execute a command in a chat room.

Precondition: The user must JOIN the room first with a call to Join Room. Otherwise you'll receive HTTP Status Code PreconditionFailed (412)

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#45c88ff5-4006-491a-b4d3-5f2ad542fa09

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val executeChatCmdResponse = withContext(Dispatchers.IO) {
        chatClient.executeChatCommand(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = ExecuteChatCommandRequest(
                command = "This is a test annoncement!",
                userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
                eventtype = "announcement"
            )
        )
    }
```

(continues on next page)

(continued from previous page)

```
    // Resolve `executeChatCmdResponse` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.12 Execute Dance Action

Invoke this function to execute a command `High five or Dance Action` in a chat room.

Precondition: The user must JOIN the room first with a call to Join Room. Otherwise you'll receive HTTP Status Code PreconditionFailed (412)

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#45c88ff5-4006-491a-b4d3-5f2ad542fa09

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val executeChatCmdResponse = withContext(Dispatchers.IO) {
        chatClient.executeChatCommand(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = ExecuteChatCommandRequest(
                command = "/high5 georgew",
                userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `executeChatCmdResponse` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.13 Reply to a Message (Threaded)

Invoke this function to create a threaded reply to another message event.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#d54ce72a-1a8a-4230-b950-0d1b345c20c6

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val sendThreadedReplyResponse = withContext(Dispatchers.IO) {
        chatClient.sendThreadedReply(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            replyTo = "0976280012ac00023242",   // ID of an existing event from this␣
↪chatroom, which you intend to reply to
            request = SendThreadedReplyRequest(
                body = "This is Jessy, replying to your greetings yow!!!",
                userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `sendThreadedReplyResponse` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.14 Quote a Message

Invoke this function to quote an existing message and republishes it with a new message.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#c463cddd-c247-4e7c-8280-2d4880813149

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
```

(continues on next page)

(continued from previous page)

```
    )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val sendQuotedReplyResponse = withContext(Dispatchers.IO) {
        chatClient.sendQuotedReply(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            replyTo = "0976280012ac00023242",   // ID of an existing event from this␣
↪chatroom, which you intend to reply to
            request = SendQuotedReplyRequest(
                body = "This is Jessy, quoting your greetings yow!!!",
                userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `sendQuotedReplyResponse` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.15 React To A Message (“Like”)

Invoke this function to add or remove a reaction to an existing event.

After this completes, a new event appears in the stream representing the reaction. The new event will have an updated version of the event in the replyto field, which you can use to update your UI.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#977044d8-9133-4185-ac1f-4d96a40aa60b

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val reactToAMsgResponse = withContext(Dispatchers.IO) {
        chatClient.reactToEvent(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            eventId = "0976280012ac00023242",   // ID of an existing event from this␣
↪chatroom, which you intend to reply to
```

(continues on next page)

(continued from previous page)

```
            request = ReactToAMessageRequest(
                userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
                reaction = "like",
                reacted = true
            )
        )
    }

    // Resolve `reactToAMsgResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.16 Report Message

Invoke this function to REPORT a message to the moderation team.

After this completes, a new event appears in the stream representing the reaction. The new event will have an updated version of the event in the replyto field, which you can use to update your UI.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#2b231a1e-a12b-4a2e-b7f3-7104bec91a0a

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val reportMsgResponse = withContext(Dispatchers.IO) {
        chatClient.reportMessage(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            eventId = "0976280012ac00023242",   // ID of an existing event from this␣
↪chatroom, which you intend to reply to
            request = ReportMessageRequest(
                reporttype = "abuse",
                userid = "023976080242ac120002" // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `reportMsgResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.17 Execute Admin Command (*help)

Invoke this function to execute `help` command in a chat room.

Precondition: The user must JOIN the room first with a call to Join Room. Otherwise you'll receive HTTP Status Code PreconditionFailed (412)

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#08b0ab21-0e9f-40a3-bdfe-f228196fea03

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val executeChatCmdResponse = withContext(Dispatchers.IO) {
        chatClient.executeChatCommand(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = ExecuteChatCommandRequest(
                command = "*help*",
                userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
            )
        )
    }

    // Resolve `executeChatCmdResponse` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.18 Get Updates

Invoke this function to get the recent updates to a room.

- You can use this function to poll the room to get the recent events in the room. The recommended poll interval is 500ms. Each event has an ID and a timestamp. To detect new messages using polling, call this function and then process items with a newer timestamp than the most recent one you have already processed.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#be93067d-562e-41b2-97b2-b2bf177f1282

Below is a code sample on how to use this SDK feature:

5.18.1 Using Coroutines Flow extension

```
import com.sportstalk.api.polling.coroutines.allEventUpdates
// ...

// Under Fragment class
// ...
// User must first Join Chat Room
// Now that the test user has joined the room, setup reactive subscription to event␣
↪updates
// Below returns a Flow<List<ChatEvent>>
lifecycleScope.launch {
    chatClient.allEventUpdates(
        chatRoomId = testChatRoom.id!!,
        lifecycleOwner = viewLifecycleOwner /* Already provided by androidx.Fragment␣
↪*/,
        frequency = 1000L /* Polling Frequency. Defaults to 500 milliseconds if not␣
↪explicitly provided */,
        /*
        * The following are placeholder/convenience functions should the developers␣
↪want to implement it
        * in a callback-oriented way. (Invoked as subscription's side-effect. In␣
↪RxJava, these are invoked via .doOnNext { ... }. In coroutine flow, these are␣
↪invoked via .onEach { ... })
        */
        onChatEvent = { event: ChatEvent -> /* Handle all other eventtype */ }, //␣
↪OPTIONAL
        onGoalEvent = { goalEvent: ChatEvent -> /* Handle eventtype == "goal" */ }, //
↪ OPTIONAL
        onAdEvent = { adEvent: ChatEvent -> /* Handle eventtype == "advertisement" */␣
↪}, // OPTIONAL
        onReply = { replyEvent: ChatEvent -> /* Handle eventtype == "reply" */ }, //␣
↪OPTIONAL
        onReaction = { reactionEvent: ChatEvent -> /* Handle eventtype == "reaction"␣
↪*/ }, // OPTIONAL
        onPurgeEvent = { purgeEvent: ChatEvent -> /* Handle eventtype == "purge" */ }␣
↪// OPTIONAL
    )
    .distinctUntilChanged()
    .onEach { events ->
        // Alternatively, the developer can opt to consume the events in here...
        // NOTE:: ONLY choose 1 approach to avoid handling it twice.
        // Iterate each event item(s)
        events.forEach { chatEvent ->
            when(chatEvent.eventtype) {
                EventType.GOAL -> { /* Handle goal event types */ }
                EventType.ADVERTISEMENT -> { /* Handle advertisements event types */ }
                // ...
                // ...
            }
        }
    }
    .launchIn(lifecycleScope /* Already provided by androidx.Fragment */)

    // Then, perform start listening to event updates
    chatClient.startListeningToChatUpdates(
        forRoomId = testChatRoom.id!!
```

(continues on next page)

(continued from previous page)

```
    )

    // At some point in time, the developer might want to explicitly stop listening␣
↪to event updates
    chatClient.stopListeningToChatUpdates(
        forRoomId = testChatRoom.id!!
    )

}
```

5.18.2 Using Rx2Java extension

```
import com.sportstalk.api.polling.rxjava.allEventUpdates
// ...

// Under Fragment class
// ...
// User must first Join Chat Room
// Now that the test user has joined the room, setup reactive subscription to event␣
↪updates
// Below returns a Flowable<List<ChatEvent>>
lifecycleScope.launch {
    chatClient.allEventUpdates(
        chatRoomId = testChatRoom.id!!,
        lifecycleOwner = viewLifecycleOwner /* Already provided by androidx.Fragment␣
↪*/,
        frequency = 1000L /* Polling Frequency. Defaults to 500 milliseconds if not␣
↪explicitly provided */,
        /*
        * The following are placeholder/convenience functions should the developers␣
↪want to implement it
        * in a callback-oriented way. (Invoked as subscription's side-effect. In␣
↪RxJava, these are invoked via .doOnNext { ... }. In coroutine flow, these are␣
↪invoked via .onEach { ... })
        */
        onChatEvent = { event: ChatEvent -> /* Handle all other eventtype */ }, //␣
↪OPTIONAL
        onGoalEvent = { goalEvent: ChatEvent -> /* Handle eventtype == "goal" */ }, //
↪ OPTIONAL
        onAdEvent = { adEvent: ChatEvent -> /* Handle eventtype == "advertisement" */␣
↪}, // OPTIONAL
        onReply = { replyEvent: ChatEvent -> /* Handle eventtype == "reply" */ }, //␣
↪OPTIONAL
        onReaction = { reactionEvent: ChatEvent -> /* Handle eventtype == "reaction"␣
↪*/ }, // OPTIONAL
        onPurgeEvent = { purgeEvent: ChatEvent -> /* Handle eventtype == "purge" */ }␣
↪// OPTIONAL
   )
   .distinctUntilChanged()
   .subscribeOn(Schedulers.io())
   .observeOn(AndroidSchedulers.mainThread())
   .subscribe { events ->
      // Alternatively, the developer can opt to consume the events in here...
      // NOTE:: ONLY choose 1 approach to avoid handling it twice.
      // Iterate each event item(s)
```

(continues on next page)

(continued from previous page)

```
      events.forEach { chatEvent ->
         when(chatEvent.eventtype) {
            EventType.GOAL -> { /* Handle goal event types */ }
            EventType.ADVERTISEMENT -> { /* Handle advertisements event types */ }
            // ...
            // ...
         }
      }
   }
   .addTo(rxDisposeBag) // RxKotlin `Observable<*>.addTo()`/`Flowable<*>.addTo()`

    // Then, perform start listening to event updates
    chatClient.startListeningToChatUpdates(
        forRoomId = testChatRoom.id!!
    )

    // At some point in time, the developer might want to explicitly stop listening␣
↪to event updates
    chatClient.stopListeningToChatUpdates(
        forRoomId = testChatRoom.id!!
    )

}
```

5.18.3 Using LiveData extension

```
import com.sportstalk.api.polling.livedata.allEventUpdates
// ...

// Under Fragment class
// ...
// User must first Join Chat Room
// Now that the test user has joined the room, setup reactive subscription to event␣
↪updates
// Below returns a LiveData<List<ChatEvent>>
lifecycleScope.launch {
    chatClient.allEventUpdates(
        chatRoomId = testChatRoom.id!!,
        lifecycleOwner = viewLifecycleOwner /* Already provided by androidx.Fragment␣
↪*/,
        frequency = 1000L /* Polling Frequency. Defaults to 500 milliseconds if not␣
↪explicitly provided */,
        /*
        * The following are placeholder/convenience functions should the developers␣
↪want to implement it
        * in a callback-oriented way. (Invoked as subscription's side-effect. In␣
↪RxJava, these are invoked via .doOnNext { ... }. In coroutine flow, these are␣
↪invoked via .onEach { ... })
        */
        onChatEvent = { event: ChatEvent -> /* Handle all other eventtype */ }, //␣
↪OPTIONAL
        onGoalEvent = { goalEvent: ChatEvent -> /* Handle eventtype == "goal" */ }, //
↪ OPTIONAL
        onAdEvent = { adEvent: ChatEvent -> /* Handle eventtype == "advertisement" */␣
↪}, // OPTIONAL
```

(continues on next page)

(continued from previous page)

```
        onReply = { replyEvent: ChatEvent -> /* Handle eventtype == "reply" */ }, //␣
↪OPTIONAL
        onReaction = { reactionEvent: ChatEvent -> /* Handle eventtype == "reaction"␣
↪*/ }, // OPTIONAL
        onPurgeEvent = { purgeEvent: ChatEvent -> /* Handle eventtype == "purge" */ }␣
↪// OPTIONAL
    )
    .distinctUntilChanged()  // livedata-ktx
    .observe(viewLifeCycleOwner, Observer { events ->
       // Alternatively, the developer can opt to consume the events in here...
       // NOTE:: ONLY choose 1 approach to avoid handling it twice.
       // Iterate each event item(s)
       events.forEach { chatEvent ->
          when(chatEvent.eventtype) {
             EventType.GOAL -> { /* Handle goal event types */ }
             EventType.ADVERTISEMENT -> { /* Handle advertisements event types */ }
             // ...
             // ...
          }
       }
    })

     // Then, perform start listening to event updates
     chatClient.startListeningToChatUpdates(
         forRoomId = testChatRoom.id!!
     )

     // At some point in time, the developer might want to explicitly stop listening␣
↪to event updates
     chatClient.stopListeningToChatUpdates(
         forRoomId = testChatRoom.id!!
     )

}
```

5.19 List Messages By User

Invoke this function to get a list of users messages.

This method requires authentication.

The purpose of this method is to get a list of messages or comments by a user, with count of replies and reaction data. This way, you can easily make a screen in your application that shows the user a list of their comment contributions and how people reacted to it.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#0ec044c6-a3c0-478f-985a-156f6f5b660a

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
    config = ClientConfig(
        appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
        apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
```

(continues on next page)

(continued from previous page)

```
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val listUserMessages = withContext(Dispatchers.IO) {
        chatClient.listMessagesByUser(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            userid = "023976080242ac120002", // ID of an existing user from this␣
↪chatroom
            limit = 20, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of user␣
↪message(s).
        )
    }

    // Resolve `listUserMessages` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.20 List Event History

Invoke this function to list events history.

- This method enables you to download all of the events from a room in large batches. It should only be used if doing a data export.
- This method returns a list of events sorted from oldest to newest.
- This method returns all events, even those in the inactive state.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#b8ca9766-ab07-4c8c-8e25-002a24a8feaa

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
```

(continues on next page)

(continued from previous page)

```
    val listEventsHistory = withContext(Dispatchers.IO) {
        chatClient.listEventsHistory(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            limit = 20, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of events.
        )
    }

    // Resolve `listEventsHistory` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.21 List Previous Events

Invoke this function to list previous events.

- This method allows you to go back in time to "scroll" in reverse through past messages. The typical use case for this method is to power the scroll-back feature of a chat window allowing the user to look at recent messages that have scrolled out of view. It's intended use is to retrieve small batches of historical events as the user is scrolling up.
- This method returns a list of events sorted from newest to oldest.
- This method excludes events that are not in the active state (for example if they are removed by a moderator)
- This method excludes non-displayable events (reaction, replace, remove, purge)
- This method will not return events that were emitted and then deleted before this method was called

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#f750f610-5db8-46ca-b9f7-a800c2e9c94a

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val listPreviousEvents = withContext(Dispatchers.IO) {
        chatClient.listPreviousEvents(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            limit = 20, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of events.
        )
```

(continues on next page)

(continued from previous page)

```
    }

    // Resolve `listPreviousEvents` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.22 Get Event by ID (TODO::)

Invoke this function to get a chat event by ID.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#04f8f563-eacf-4a64-9f00-b3d6c050a2fa

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // TODO:: NOT YET IMPLEMENTED
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val chatEventResponse = withContext(Dispatchers.IO) {
        chatClient.getChatEventById(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            chatEventId = "7620812242ac09300002"    // ID of an existing event from␣
↪the chat room
        )
    }

    // Resolve `chatEventResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.23 Purge User Messages

Invoke this function to execute a command in a chat room to purge all messages for a user.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#04ffee45-a3e6-49b8-8968-46b219020b66

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val purgeCmdResponse = withContext(Dispatchers.IO) {
        chatClient.executeChatCommand(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            // Assuming ADMIN password "testpassword123"
            // Assuming user "@nicoleWd" exists
            request = ExecuteChatCommandRequest(
                command = "*purge testpassword123 nicoleWd",
                userid = "023976080242ac120002" // ID of an existing user "@nicoleWd"␣
↪from this chatroom
            )
        )
    }

    // Resolve `purgeCmdResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.24 Bounce User

Invoke this function to remove the user from the room and prevent the user from reentering.

Optionally display a message to people in the room indicating this person was bounced.

When you bounce a user from the room, the user is removed from the room and blocked from reentering. Past events generated by that user are not modified (past messages from the user are not removed).

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#7116d7ca-a1b8-44c1-8894-bea85225e4c7

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)
```

(continues on next page)

(continued from previous page)

```
// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val bounceUserResponse = withContext(Dispatchers.IO) {
        chatClient.bounceUser(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            // Assuming user "@nicoleWd" exists
            request = BounceUserRequest(
                userid = "023976080242ac120002", // ID of an existing user "@nicoleWd
↪" from this chatroom
                bounce = true,
                announcement = "@nicoleWd has been banned."
            )
        )
    }

    // Resolve `bounceUserResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.25 Flag Message Event As Deleted

Invoke this function to set a ChatEvent as logically deleted.

Everything in a chat room is an event. Each event has a type. Events of type "speech, reply, quote" are considered "messages".

Use logical delete if you want to flag something as deleted without actually deleting the message so you still have the data. When you use this method:

- The message is not actually deleted. The comment is flagged as deleted, and can no longer be read, but replies are not deleted.
- If flag "permanentifnoreplies" is true, then it will be a permanent delete instead of logical delete for this comment if it has no children.
- If you use "permanentifnoreplies" = true, and this comment has a parent that has been logically deleted, and this is the only child, then the parent will also be permanently deleted (and so on up the hierarchy of events).

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#92632caf-9bd0-449d-91df-90fef54f6634

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)
```

(continues on next page)

(continued from previous page)

```
// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val deleteEventResponse = withContext(Dispatchers.IO) {
        chatClient.flagEventLogicallyDeleted(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            eventId = "7620812242ac09300002",    // ID of an existing event from the␣
↪chat room
            userid = "023976080242ac120002", // ID of an existing user "@nicoleWd"␣
↪from this chatroom
            // Assuming user "@nicoleWd" exists
            permanentifnoreplies = true
        )
    }

    // Resolve `deleteEventResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.26 Delete Event

Invoke this function to deletes an event from the room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#f2894c8f-acc9-4b14-a8e9-216b28c319de

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val removeEventResponse = withContext(Dispatchers.IO) {
        chatClient.removeEvent(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            eventId = "7620812242ac09300002",    // ID of an existing event from the␣
↪chat room
            userid = "023976080242ac120002", // ID of an existing user "@nicoleWd"␣
↪from this chatroom
        )
    }
```

(continues on next page)

(continued from previous page)

```
    // Resolve `removeEventResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.27 Delete All Events in Room

Invoke this function to execute a command in a chat room to delete all messages in the chatroom.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#e4d62330-469e-4e37-a42e-049b10259152

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val deleteAllEventsCmdResponse = withContext(Dispatchers.IO) {
        chatClient.executeChatCommand(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            // Assuming ADMIN password "testpassword123"
            request = ExecuteChatCommandRequest(
                command = "*deleteallevents testpassword123",
                userid = "023976080242ac120002" // ID of an existing user "@nicoleWd"␣
↪from this chatroom
            )
        )
    }

    // Resolve `deleteAllEventsCmdResponse` from HERE onwards(ex. update UI␣
↪displaying the response data)...
}
```

5.28 Update Room (Close a room)

Invoke this function to update an existing room.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#e4d62330-469e-4e37-a42e-049b10259152

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val updatedEventResponse = withContext(Dispatchers.IO) {
        chatClient.updateRoom(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            request = UpdateChatRoomRequest(
                    name = "Test Chat Room 1 - UPDATED",
                    customid = "test-room-1-updated",
                    description = "[UPDATED] This is a test chat room 1.",
                    moderation = "post",
                    enableactions = false,
                    enableenterandexit = false,
                    enableprofanityfilter = true,
                    delaymessageseconds = 10L,
                    roomisopen = false,
                    maxreports = 30
            )
        )
    }

    // Resolve `updatedEventResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.29 Exit a Room

Invoke this function to exit from a chatroom where the user has currently joined.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#408b43ca-fca9-4f2d-8883-f6f725d140f2

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val exitRoomResponse = withContext(Dispatchers.IO) {
        chatClient.exitRoom(
            chatRoomId = "080001297623242ac002",    // ID of an existing chat room
            userid = "023976080242ac120002" // ID of an existing user from this␣
↪chatroom
        )
    }

    // Resolve `exitRoomResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.30 Delete Room

Invoke this function to delete the specified room and all events contained therein) by ID

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#c5ae345d-004d-478a-b543-5abaf691000d

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val deleteRoomResponse = withContext(Dispatchers.IO) {
        chatClient.deleteRoom(
            chatRoomId = "080001297623242ac002"    // ID of an existing chat room
        )
    }
```

(continues on next page)

(continued from previous page)

```
    // Resolve `deleteRoomResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

5.31 List Messages Needing Moderation

Invoke this function to list all the messages in the moderation queue.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#bcdbda1b-e495-46c9-8fe9-c5dc6a4c1756

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val listMessagesInModeration = withContext(Dispatchers.IO) {
        chatClient.listMessagesNeedingModeration(
            roomId = "080001297623242ac002",    // ID of an existing chat room
            limit = 20, /* Defaults to 200 on backend API server */
            cursor = null // OPTIONAL: The cursor value from previous search attempt␣
↪to indicate next paginated fetch. Null if fetching the first list of messages from␣
↪this chatroom.
        )
    }

    // Resolve `listMessagesInModeration` from HERE onwards(ex. update UI displaying␣
↪the response data)...
}
```

5.32 Approve Message

Invoke this function to approve a message in the moderation queue.

Refer to the SportsTalk API Documentation for more details:

https://apiref.sportstalk247.com/?version=latest#6f9bf714-5b3b-48c9-87d2-eb2e12d2bcbf

Below is a code sample on how to use this SDK feature:

```
val chatClient = SportsTalk247.ChatClient(
   config = ClientConfig(
      appId = "c84cb9c852932a6b0411e75e", // This is just a sample app id
      apiToken = "5MGq3XbsspBEQf3kj154_OSQV-jygEKwHJyuHjuAeWHA", // This is just a␣
↪sample token
      endpoint = "http://api.custom.endpoint/v1/" // This is just a sample API␣
↪endpoint
   )
)

// Launch thru coroutine block
// https://developer.android.com/topic/libraries/architecture/coroutines
lifecycleScope.launch {
    // Switch to IO Coroutine Context(Operation will be executed on IO Thread)
    val approveResponse = withContext(Dispatchers.IO) {
        chatClient.approveMessage(
            eventId = "0976280012ac00023242",   // ID of an existing event from this␣
↪chatroom, which you intend to reply to
            approve = true
        )
    }

    // Resolve `approveResponse` from HERE onwards(ex. update UI displaying the␣
↪response data)...
}
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search