

---

# **Sportstalk SDK -iOS**

**Theodore Angelo Lesano**

**Mar 02, 2023**



## CONTENTS

<b>1</b>	<b>GETTING STARTED: Setting up the SDK</b>	<b>3</b>
<b>2</b>	<b>Implement Custom JWT</b>	<b>5</b>
<b>3</b>	<b>Callback Function Overview</b>	<b>7</b>
<b>4</b>	<b>Creating/Updating a user</b>	<b>9</b>
<b>5</b>	<b>Joining a Room</b>	<b>11</b>
<b>6</b>	<b>Joining a Room using Custom ID</b>	<b>13</b>
<b>7</b>	<b>Getting room updates</b>	<b>15</b>
<b>8</b>	<b>Start/Stop Getting Event Updates</b>	<b>17</b>
<b>9</b>	<b>Sending A Message</b>	<b>19</b>
<b>10</b>	<b>Conversations and Comments</b>	<b>21</b>
<b>11</b>	<b>The Bare Minimum</b>	<b>23</b>
<b>12</b>	<b>Chat Application Best Practices</b>	<b>25</b>
<b>13</b>	<b>User Client</b>	<b>27</b>
13.1	Create/Update User . . . . .	27
13.2	Delete User . . . . .	28
13.3	Get User Details . . . . .	29
13.4	List Users . . . . .	30
13.5	Ban/Unban User . . . . .	30
13.6	Global Purge User . . . . .	31
13.7	Search User . . . . .	32
13.8	Mute User . . . . .	32
13.9	Report User . . . . .	33
13.10	Shadow Ban User . . . . .	34
13.11	List User Notifications . . . . .	35
13.12	Mark All Notification As Read . . . . .	36
13.13	Set User Notification As Read . . . . .	37
13.14	Set User Notification As Read (By ChatEventId) . . . . .	38
13.15	Delete User Notification . . . . .	39
13.16	Delete User Notification By ChatEventId . . . . .	40

<b>14 Chat Client</b>	<b>41</b>
14.1 Create Room . . . . .	41
14.2 Get Room Details . . . . .	42
14.3 Get Room Extended Details . . . . .	43
14.4 Get Room Details By Custom ID . . . . .	45
14.5 Delete Room . . . . .	46
14.6 Update Room . . . . .	46
14.7 Update and Close Room . . . . .	48
14.8 List Rooms . . . . .	49
14.9 List Room Participants . . . . .	50
14.10 List User Subscribed Rooms . . . . .	50
14.11 List Event History . . . . .	51
14.12 List Previous Events . . . . .	52
14.13 List Event By Type . . . . .	53
14.14 List Event By Timestamp . . . . .	54
14.15 Join Room . . . . .	55
14.16 Join Room by CustomId . . . . .	56
14.17 Exit Room . . . . .	58
14.18 Get Updates . . . . .	59
14.19 Get More Updates . . . . .	60
14.20 Execute Command . . . . .	61
14.21 Send Quoted Reply . . . . .	63
14.22 Send Threaded Reply . . . . .	64
14.23 List Messages By User . . . . .	66
14.24 Purge Message . . . . .	66
14.25 Flag Event As Locally Deleted . . . . .	67
14.26 Permanently Delete Event . . . . .	68
14.27 Delete All Events . . . . .	69
14.28 List Messages of User . . . . .	69
14.29 Report A Message . . . . .	70
14.30 React to an Event . . . . .	71
14.31 Report User in Room . . . . .	73
14.32 Bounce User . . . . .	74
14.33 Shadowban User . . . . .	75
14.34 Mute User . . . . .	76
14.35 Search Event History . . . . .	77
14.36 Update Chat Event . . . . .	79
14.37 Start Listening to Chat Updates . . . . .	80
14.38 Stop Listening to Chat Updates . . . . .	81
14.39 Approve Event . . . . .	82
14.40 Reject Event . . . . .	83
14.41 List All Messages In Moderation Queue . . . . .	84
<b>15 Comment Client</b>	<b>85</b>
15.1 Create or Update Conversation . . . . .	85
15.2 Get Conversation by ID . . . . .	87
15.3 Find Conversation by CustomID . . . . .	88
15.4 List Conversations . . . . .	90
15.5 Batch Get Conversation Details . . . . .	91
15.6 React to Conversation Topic . . . . .	92
15.7 Create and Publish Comment . . . . .	94
15.8 Reply to Comment . . . . .	96
15.9 List Replies . . . . .	98
15.10 Get Comment by ID . . . . .	100

15.11 List Comments . . . . .	101
15.12 List Replies Batch . . . . .	103
15.13 React to Comment(“Like”) . . . . .	104
15.14 Vote on Comment . . . . .	106
15.15 Report Comment . . . . .	108
15.16 Update Comment . . . . .	109
15.17 Flag Comment As Deleted . . . . .	111
15.18 Delete Comment (permanent) . . . . .	112
15.19 Delete Conversation . . . . .	113
15.20 List Comments in Moderation Queue . . . . .	114
15.21 Approve/Reject Message in Queue . . . . .	115
<b>16 Copyright &amp; License</b>	<b>119</b>



The Sportstalk SDK is a helpful wrapper around the [Sportstalk API](#)

The set of SDKs and source (iOS, Android, and JS) is here: <https://gitlab.com/sportstalk247/>

```
pod 'SportsTalk_iOS_SDK', :git=> 'https://gitlab.com/sportstalk247/sdk-ios-swift.git'
```

You will need to register with SportsTalk and get an API Key in order to use SDK functions.





## GETTING STARTED: SETTING UP THE SDK

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

Sportstalk is an EVENT DRIVEN API. When new talk events occur, the SDK will trigger appropriate callbacks, if set.

```
import SportsTalk_iOS_SDK

// First you'll need to create a ClientConfig class that you can use later on
let config = ClientConfig(appId: "YourAppId", authToken: "YourApiKey", endpoint: "Your_
↳ URL")
let client = UserClient(config: config)

// You can set config to have your own endpoint or use the default endpoint like so
let config = ClientConfig(appId: "YourAppId", authToken: "YourApiKey")
```



## IMPLEMENT CUSTOM JWT

You can instantiate a `JWTProvider` instance and provide a token refresh action function that returns a new token.

```
import SportsTalk_iOS_SDK

// First you'll need to create a ClientConfig class that you can use later on
let config = ClientConfig(appId: "YourAppId", authToken: "YourApiKey", endpoint: "Your_
↳URL")
// Prepare JWTProvider
let jwtProvider = JWTProvider(
    tokenRefreshFunction: { completion in
        DispatchQueue.main.async {
            let newToken = doPerformFetchNewToken() // Developer may perform a long-
↳running operation to generate a new JWT
            completion(newToken)
        }
    }
)
// Set custom JWTProvider
SportsTalkSDK.setJWTProvider(config: config, provider: jwtProvider)
```

You can also directly specify the JWT value by calling `JWTProvider.setToken(newToken)`. There is also a function provided to explicitly refresh token by calling `JWTProvider.refreshToken()`, which will trigger the provided token refresh action above to fetch a new token and will automatically add that on the SDK.

```
// Continuation from above

client.createRoom(request) { (code, message, kind, room) in
    // ...
    // Handle Unauthorized Error
    // - Attempt request refresh token
    //
    if code == 401 {
        jwtProvider.refreshToken()
        // Then, probably prompt for another retry attempt again after a_
↳shortwhile(this is to ensure that the token gets refreshed first before retry_
↳attempt)
    }
}
```

Once the User Token has been added to the SDK, the SDK will automatically append it to all requests.



## **CALLBACK FUNCTION OVERVIEW**

Each and every api function has its callback, when the api is called you will get the response in the callback. You can use this to remove loading screens, hide advertisements, and so on.



## CREATING/UPDATING A USER

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

When users send messages to a room the user ID is passed as a parameter. When you retrieve the events from a room, the user who generated the event is returned with the event data, so it is easy for your application to process and render chat events with minimal code.

```
import SportsTalk_iOS_SDK

let client = UserClient(config: config)

// Almost all api is designed to have a request and response model.

func createUser() {
    // To create a request, make use of the Services convenience class
    let request = UserRequest.CreateUpdateUser()
    request.userid = "SomeUserId"
    request.handle = "Sam"
    request.displayname = "Sam"
    request.pictureurl = URL(string: <some_url>)
    request.profileurl = URL(string: <some_url>)

    client.createOrUpdateUser(request) { (code, message, kind, user) in
        // where; code: Int?, message: String?, kind: String?, user: User?
        // Save user
    }
}
```





## JOINING A ROOM

```
let client = ChatClient(config: config)

func JoinRoom(_ room: ChatRoom, as user: User) {
    let request = ChatRequest.JoinRoom()
    request.roomid = room.id
    request.userid = user.userid
    request.displayname = user.displayname

    client.joinRoom(request) { (code, message, _, response) in
        // where response is model JoinChatRoomResponse
        // Process response
    }
}
```



## JOINING A ROOM USING CUSTOM ID

```
let client = ChatClient(config: config)

func JoinRoom(_ room: ChatRoom, as user: User) {
    let request = ChatRequest.JoinRoomByCustomId()
    request.userid = user.userid
    request.displayname = user.displayname
    request.customid = room.customid

    client.joinRoomByCustomId(request) { (code, message, _, response) in
        // where response is model called JoinChatRoomResponse
        // Process response
    }
}
```



## GETTING ROOM UPDATES

To manually get room updates, use `ChatClient().getUpdates(request:completionHandler)`

```
let client = ChatClient(config: config)

func getUpdates(_ room: ChatRoom) {
    let request = ChatRequest.GetUpdates()
    request.roomid = room.id
    request.limit = 20

    client.getUpdates(request) { (code, message, _, response) in
        // where response is model called GetUpdatesResponse
        // Get an array of events from response.events
    }
}
```



## START/STOP GETTING EVENT UPDATES

Get periodic updates from room by using `func startListeningToChatUpdates(config: ChatRequest.StartListeningToChatUpdates, completionHandler: @escaping Completion<[Event]>)` Only new events will be emitted, so it is up to you to collect the new events. To stop getting updates, simply call `client.stopListeningToChatUpdates()` anytime.

Note: Updates are received every 500 milliseconds. You can configure the delivery of messages by setting `ChatRequest.StartListeningToChatUpdates` Losing reference to client will stop the eventUpdates

```
let client = ChatClient(config: config)
var events = [Event]()

func receiveUpdates(from room: ChatRoom) {
    let eventUpdatesConfig = ChatRequest.StartListeningToChatUpdates(roomid: room.id!)
    client.startListeningToChatUpdates(config: eventUpdatesConfig) { (code, message, _, ↵
↵event) in
        if let event = event {
            events.append(event)
        }

        // Debug pulse
        print("-----")
        print(code == 200 ? "pulse success" : "pulse failed")
        print((event?.count ?? 0) > 0 ? "received \(event?.count) event" : "No new events
↵")
        print("-----")
        receivedCode = code
    }
}

func stopUpdates(from room: ChatRoom) {
    // Ideally call this on viewDidDisappear() and deinit()
    let roomid = room.id!
    client.stopListeningToChatUpdates(roomid)
}
```





## SENDING A MESSAGE

Use SAY command to send a message to the room.

example: SAY Hello World! or simply Hello World!

Perform ACTIONS by using / character

example: /dance nicole

- User sees: You dance with Nicole
- Nicole sees: (user) dances with you
- Everyone else sees: (user) 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

```
let client = ChatClient(config: config)

func send(message: String, to room: ChatRoom, as user: User) {
    // See for list of commands

    do {
        let request = ChatRequest.ExecuteChatCommand()
        request.roomId = room.id
        request.command = "SAY \(message)"
        request.userid = user.userid

        client.executeChatCommand(request) { (code, message, _, response) in
            // where response is model ExecuteChatCommandResponse
            // Process response
        }
    } catch {
        // Handle errors
    }
}
```

For use of these events in action, see the demo page: <https://www.sportstalk247.com/demo.html>



## CONVERSATIONS AND COMMENTS

```
let client = CommentClient(config: config)

func getConversations() {
    let request = CommentRequest.ListConversations()

    client.listConversations(request) { (code, message, _, response) in
        // where response is model called ListConversationsResponse
        // Get an array of conversations from response.conversations
    }
}
```



## THE BARE MINIMUM

The only critical events that you need to handle are `ExecuteChatCommand` which will be called for each new chat event and `PurgeMessage` which will be called when purge commands are issued to clear messages that violate content policy.

You will probably also want to use `ExecuteChatCommand` to show/hide any loading messages.

The easiest way to see how these event works is to see the demo page: <https://www.sportstalk247.com/demo.html>



## CHAT APPLICATION BEST PRACTICES

Do not ‘fire and forget’ chat messages. Most chat applications require some level of moderation. Your UI should make sure to keep track of message metadata such as:

- Message ID
- User Handle for each message.
- User ID for each message. In the event of moderation or purge events, your app will need to be able to find and remove purged messages.
- Timestamp

Make sure you handle errors for sending messages in case of network disruption.

Enable/Disable debug mode with `SportsTalkSDK.shared.debugMode = true/false`





## USER CLIENT

### 13.1 Create/Update User

```
func createOrUpdateUser(_ request: UserRequest.CreateUpdateUser, completionHandler: @escaping Completion<User>)
```

All users must have a Handle. The display name is optional. If you create a user and don't provide a handle, but you do provide a display name, a handle will be generated for you based on the provided display name. The generated handle will not be able to contain all characters or spaces, and could have numbers appended to the end.

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

Do not use this method to convert an anonymous user into a known user. Use the Convert User api method instead.

When users send messages to a room the user ID is passed as a parameter. When you retrieve the events from a room, the user who generated the event is returned with the event data, so it is easy for your application to process and render chat events with minimal code.

#### Parameters

- **userid:** (required) If the userid is new then the user will be created. If the userid is already in use in the database then the user will be updated.
- **handle:** (optional) A unique string representing the user that is easy for other users to type. Example @George-Washington could be the handle but Display Name could be "Wooden Teef For The Win".
- **displayname:** (optional) This is the desired name to display, typically the real name of the person.
- **pictureurl:** (optional) The URL to the picture for this user.
- **profileurl:** (optional) The profileurl for this user.

#### Note about handles

- If you are creating a user and you don't specify a handle, the system will generate one for you (using Display Name as basis if you provide that).
- If you request a handle and it's already in use a new handle will be generated for you by adding a number from 1-99 and returned.
- If the handle can't be generated because all the options 1-99 on the end of it are taken then the request will be rejected with BadRequest status code.
- **Only these characters may be used:**  
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\_"

**Request Model:** UserRequest.CreateUpdateUser

```
public class CreateUser {
    public var userid: String?
    public var handle: String?
    public var displayname: String?
    public var pictureurl: URL?
    public var profileurl: URL?
}
```

**Response Model: User**

```
open class User: NSObject, Codable {
    public var kind: String?
    public var userid: String?
    public var handle: String?
    public var profileurl: String?
    public var banned: Bool?
    public var banexpires: Date?
    public var shadowbanned: Bool?
    public var shadowbanexpires: Date?
    public var muted: Bool?
    public var muteexpires: Date?
    public var moderation: String?
    public var displayname: String?
    public var handlelowercase: String?
    public var pictureurl: String?
    public var reports: [UserReport]?
    public var role: Role?
    public var customtags: [String]?
}
```

## 13.2 Delete User

```
func deleteUser(_ request: UserRequest.DeleteUser, completionHandler: @escaping _
↳ Completion<DeleteUserResponse>)
```

Deletes the specified user.

All rooms with messages by that user will have the messages from this user purged in the rooms.

**Parameters**

- userid: (required) is the app specific User ID provided by your application.

**Warning** This method requires authentication

**Request Model:** UserRequest.DeleteUser

```
public class DeleteUser {
    public var userid: String?
}
```

**Response Model:** DeleteUserResponse

```
public struct DeleteUserResponse: Codable {
    public var kind: String?
    public var user: User?
}
```

## 13.3 Get User Details

```
func getUserDetails(_ request: UserRequest.GetUserDetails, completionHandler: @escaping _
↳ Completion<User>)
```

Get the details about a User.

This will return all the information about the user.

### Parameters

- userid: (required) is the app specific User ID provided by your application.

**Warning** This method requires authentication

**Request Model:** `UserRequest.GetUserDetails`

```
public class GetUserDetails {
    public var userid: String?
}
```

**Response Model:** `User`

```
open class User: NSObject, Codable {
    public var kind: String?
    public var userid: String?
    public var handle: String?
    public var profileurl: String?
    public var banned: Bool?
    public var banexpires: Date?
    public var shadowbanned: Bool?
    public var shadowbanexpires: Date?
    public var muted: Bool?
    public var muteexpires: Date?
    public var moderation: String?
    public var displayname: String?
    public var handlelowercase: String?
    public var pictureurl: String?
    public var reports: [UserReport]?
    public var role: Role?
    public var customtags: [String]?
}
```

## 13.4 List Users

```
func listUsers(_ request: UserRequest.ListUsers, completionHandler: @escaping Completion
    ↳<ListUsersResponse>)
```

Gets a list of users.

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

### Parameters

- cursor: (optional) Each call to ListUsers will return a result set with a 'nextCursor' value. To get the next page of users, pass this value as the optional 'cursor' property. To get the first page of users, omit the 'cursor' argument.
- limit: (optional) You can omit this optional argument, in which case the default limit is 200 users to return.

**Warning** This method requires authentication

**Request Model:** UserRequest.ListUsers

```
public class ListUsers {
    public var cursor: String?
    public var limit: Int? = 200
}
```

**Response Model:** ListUsersResponse

```
public struct ListUsersResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var users: [User]
}
```

## 13.5 Ban/Unban User

```
func setBanStatus(_ request: UserRequest.SetBanStatus, completionHandler: @escaping
    ↳Completion<User>)
```

Will toggle the user's banned flag.

### Parameters

- userid: (required) The applicaiton provided userid of the user to ban
- banned: (required) Boolean. If true, user will be set to banned state. If false, wbe set to non-banned state.

**Request Model:** UserRequest.SetBanStatus

```
public class SetBanStatus {
    public var userid: String?
    public var banned: Bool?
}
```

**Response Model:** User

```

open class User: NSObject, Codable {
    public var kind: String?
    public var userid: String?
    public var handle: String?
    public var profileurl: String?
    public var banned: Bool?
    public var banexpires: Date?
    public var shadowbanned: Bool?
    public var shadowbanexpires: Date?
    public var muted: Bool?
    public var muteexpires: Date?
    public var moderation: String?
    public var displayname: String?
    public var handlelowercase: String?
    public var pictureurl: String?
    public var reports: [UserReport]?
    public var role: Role?
    public var customtags: [String]?
}

```

## 13.6 Global Purge User

```

func globallyPurgeUserContent(_ request: UserRequest.GloballyPurgeUserContent,
    ↪completionHandler: @escaping Completion<GlobalPurgeReponse>)

```

Will purge all chat content published by the specified user

### Parameters

- userid: (required) ID of the User who's content is about to be purged
- byuserid: (required) ID of the User who is about to perform the purge action(requires admin privileges)

### Request Model: UserRequest.GloballyPurgeUserContent

```

public class GloballyPurgeUserContent {
    public var userid: String?
    public var byuserid: String?
}

```

### Response Model: GlobalPurgeReponse

```

public struct GlobalPurgeReponse: Codable {}

```

## 13.7 Search User

```
func searchUser(_ request: UserRequest.SearchUser, completionHandler: @escaping _
↳Completion<ListUsersResponse>)
```

Searches the users in an app

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

### Parameters

- cursor: (optional) Each call to ListUsers will return a result set with a 'nextCursor' value. To get the next page of users, pass this value as the optional 'cursor' property. To get the first page of users, omit the 'cursor' argument.
- limit: (optional) You can omit this optional argument, in which case the default limit is 200 users to return.
- name: (optional) Provide part of a name to search the user name field
- handle: (optional) Provide part of a handle to search by handle
- userid: (optional) Provide part of a userid to search by userid

**Note** At least one of these parameters is required; - userid - handle - name

**Warning** This method requires authentication

**Request Model:** UserRequest.SearchUser

```
public class SearchUser {
    public var cursor:String?
    public var limit:Int?
    public var name:String?
    public var handle:String?
    public var userid:String?
}
```

**Response Model:** ListUsersResponse

```
public struct ListUsersResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var users: [User]
}
```

## 13.8 Mute User

```
func muteUser(_ request: ChatRequest.MuteUser, completionHandler: @escaping Completion
↳<ChatRoom>)
```

Will toggle the user's mute effect

A muted user is in a read-only state. The muted user can join chat rooms and observe but cannot communicate. This method applies mute on the global level (applies to all talk contexts). You can optionally specify an expiration time. If the expiration time is specified, then each time the shadow banned user tries to send a message the API will check if the shadow ban has expired and will lift the ban.

**Parameters**

- **userid:** (required) The applicaiton provided userid of the user to ban
- **applyeffect:** (required) true or false. If true, user will be set to muted state. If false, will be set to non-banned state.
- **expireseconds:** (optional) Duration of mute in seconds. If specified, the mute will be lifted when this time is reached. If not specified, mute effect remains until explicitly lifted. Maximum seconds is a double byte value.

**Request Model: UserRequest.MuteUser**

```
public class MuteUser {
    public var userid: String?
    public var applyeffect: Bool?
    public var expireseconds: Double?
}
```

**Response Model: User**

```
open class User: NSObject, Codable {
    public var kind: String?
    public var userid: String?
    public var handle: String?
    public var profileurl: String?
    public var banned: Bool?
    public var banexpires: Date?
    public var shadowbanned: Bool?
    public var shadowbanexpires: Date?
    public var muted: Bool?
    public var muteexpires: Date?
    public var moderation: String?
    public var displayname: String?
    public var handlelowercase: String?
    public var pictureurl: String?
    public var reports: [UserReport]?
    public var role: Role?
    public var customtags: [String]?
}
```

## 13.9 Report User

```
func reportUser(_ request: UserRequest.ReportUser, completionHandler: @escaping _
↳ Completion<User>)
```

**Parameters**

- **userid:** (required) This is the application specific user ID of the user reporting the first user.
- **reporttype:** (required) Possible values: “abuse”, “spam”. SPAM is unsolicited commercial messages and abuse is hate speech or other unacceptable behavior.

**RESPONSE CODES**

- 200 | Success : Request completed successfully
- 404 | Not Found : The specified user or application could not be found

- 409 | Conflict : The request was rejected because user reporting is not enabled for the application

**Request Model: UserRequest.ReportUser**

```
public class ReportUser {  
    public var userid: String?  
    public var reporttype = "abuse"  
}
```

**Response Model: User**

```
open class User: NSObject, Codable {  
    public var kind: String?  
    public var userid: String?  
    public var handle: String?  
    public var profileurl: String?  
    public var banned: Bool?  
    public var banexpires: Date?  
    public var shadowbanned: Bool?  
    public var shadowbanexpires: Date?  
    public var muted: Bool?  
    public var muteexpires: Date?  
    public var moderation: String?  
    public var displayname: String?  
    public var handlelowercase: String?  
    public var pictureurl: String?  
    public var reports: [UserReport]?  
    public var role: Role?  
    public var customtags: [String]?  
}
```

## 13.10 Shadow Ban User

```
func setShadowBanStatus(_ request: UserRequest.SetShadowBanStatus, completionHandler: @escaping Completion<User>)
```

Will toggle the user's shadow banned flag

A Shadow Ban user can send messages into a chat room, however those messages are flagged as shadow banned. This enables the application to show those messages only to the shadow banned user, so that that person may not know they were shadow banned. This method shadow bans the user on the global level (or you can use this method to lift the ban). You can optionally specify an expiration time. If the expiration time is specified, then each time the shadow banned user tries to send a message the API will check if the shadow ban has expired and will lift the ban.

### Parameters

- userid: (required) The applicaiton provided userid of the user to ban
- shadowban: (required) true or false. If true, user will be set to banned state. If false, will be set to non-banned state.
- expireseconds: (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.

**Request Model: UserRequest.ReportUser**



```
public class SetShadowBanStatus {
    public var userid: String?
    public var shadowban: Bool?
    public var expireseconds: Int?
}
```

#### Response Model: User

```
open class User: NSObject, Codable {
    public var kind: String?
    public var userid: String?
    public var handle: String?
    public var profileurl: String?
    public var banned: Bool?
    public var banexpires: Date?
    public var shadowbanned: Bool?
    public var shadowbanexpires: Date?
    public var muted: Bool?
    public var muteexpires: Date?
    public var moderation: String?
    public var displayname: String?
    public var handlelowercase: String?
    public var pictureurl: String?
    public var reports: [UserReport]?
    public var role: Role?
    public var customtags: [String]?
}
```

## 13.11 List User Notifications

```
func listUserNotifications(_ request: UserRequest.ListUserNotifications,
    ↪completionHandler: @escaping Completion<ListNotificationResponse>)
```

Returns a list of user notifications

#### Parameters

- userid: (required) Return only notifications for this user
- filternotificationtypes: (optional) Return only events of the specified type. Pass the argument more than once to fetch multiple types of notifications at once.
  - chatmention
  - chatquote
  - chatreply
  - commentmention
  - commentquote
  - commentreply
- includeread: (optional | default = false) If true, notifications that have already been read are returned

- filterchatroomid: (optional) If provided, this will only return notifications associated with the specified chat room using the ChatRoom ID (exact match)
- filterchatroomcustomid: (optional) If provided, this will only return notifications associated with the specified chat room using the Custom ID (exact match)
- limit: (optional) Default is 50, maximum is 200. Limits how many items are returned.
- cursor: (optional) Leave blank to start from the beginning of the result set; provide the value from the previous returned cursor to resume cursoring through the next page of results

**Request Model: UserRequest.ListUserNotifications**

```
public class ListUserNotifications {  
    public var userid: String?  
    public var filternotificationtypes: String?  
    public var includeread: Bool? = false  
    public var filterchatroomid: String?  
    public var filterchatroomcustomid: String?  
    public var limit: Int? = 50  
    public var cursor: String? = ""  
}
```

**Response Model: ListNotificationResponse**

```
public struct ListNotificationResponse: Codable {  
    public var kind: String?  
    public var cursor: String?  
    public var more: Bool?  
    public var itemcount: Int?  
    public var notifications: [UserNotification]?  
}
```

## 13.12 Mark All Notification As Read

```
func markAllNotificationAsRead(_ request: UserRequest.MarkAllNotificationAsRead,   
↪completionHandler: @escaping Completion<UserNotification>)
```

This marks all of the user's notifications as read with one API call only. Due to caching, a call to List User Notifications may still return items for a short time. Set delete = true to delete the notification instead of marking it read. This should be used for most use cases.

### Parameters

- userid: (required) The ID of the user marking the notification as read.
- delete: (optional) [default=true] If true, this deletes the notification. If false, it marks it read but does not delete it.

**Request Model: UserRequest.MarkAllNotificationAsRead**

```
public class MarkAllNotificationAsRead {  
    public var userid: String?  
    public var delete: Bool? = true  
}
```

**Response Model: UserNotification**

```
open class UserNotification: Codable {
    public var kind: String?
    public var id: String?
    public var added: Date?
    public var userid: String?
    public var ts: Date?
    public var whenread: String?
    public var isread: Bool?
    public var notificationtype: String?
    public var chatroomid: String?
    public var chatroomcustomid: String?
    public var commentconversationid: String?
    public var commentconversationcustomid: String?
    public var chateventid: String?
    public var commentid: String?
}
```

## 13.13 Set User Notification As Read

```
func setUserNotificationAsRead(_ request: UserRequest.SetUserNotificationAsRead,
    ↪completionHandler: @escaping Completion<UserNotification>)
```

**Set User Notification as Read**

Unless your workflow must support use of read notifications, instead use `func deleteUserNotification(_ request:completionHandler:)``

This marks a notification as being in READ status. That will prevent the notification from being returned in a call to List User Notifications unless the default filters are overridden. Notifications that are marked as read will be automatically deleted after some time.

Calling this over and over again for an event, or calling it on events where the reader is not the person that the reply is directed to, or calling it against events that are not type ChatReply or ChatQuote is inappropriate use of the API

**Parameters**

- **userid:** (required) The ID of the user marking the notification as read. This is used to ensure a user can't mark another user's notification as read.
- **notificationid:** (required) The unique ID of the notification being updated
- **read:** (required) The read status (true/false) for the notification. You can pass false to mark the notification as unread

**Request Model: UserRequest.SetUserNotificationAsRead**

```
public class SetUserNotificationAsRead {
    public var userid: String?
    public var notificationid: String?
    public var read: Bool? = false
}
```

**Response Model: UserNotification**

```

open class UserNotification: Codable {
    public var kind: String?
    public var id: String?
    public var added: Date?
    public var userid: String?
    public var ts: Date?
    public var whenread: String?
    public var isread: Bool?
    public var notificationtype: String?
    public var chatroomid: String?
    public var chatroomcustomid: String?
    public var commentconversationid: String?
    public var commentconversationcustomid: String?
    public var chateventid: String?
    public var commentid: String?
}

```

## 13.14 Set User Notification As Read (By ChatEventId)

```

func setUserNotificationAsReadByEventId(_ request: UserRequest.
↳ SetUserNotificationAsReadByChatEventId, completionHandler: @escaping Completion
↳ <UserNotification>)

```

Unless your workflow must support use of read notifications, use `func deleteUserNotification(\_ request:completionHandler:)` instead.

- This marks a notification as being in READ status.
- That will prevent the notification from being returned in a call to List User Notifications unless the default filters are overridden.
- Notifications that are marked as read will be automatically deleted after some time.
- Only call this once per event. Only call this for events of type ChatReply or ChatQuote

### Parameters

- userid: (required) The ID of the user marking the notification as read. This is used to ensure a user can't mark another user's notification as read.
- chateventid: (required) The unique ID of the notification's chatEvent.
- read: (required) The read status (true/false) for the notification. You can pass false to mark the notification as unread.

### Request Model: UserRequest.SetUserNotificationAsReadByChatEventId

```

public class SetUserNotificationAsReadByChatEventId {
    public var userid: String?
    public var eventid: String?
    public var read: Bool? = false
}

```

### Response Model: UserNotification

```

open class UserNotification: Codable {
    public var kind: String?
    public var id: String?
    public var added: Date?
    public var userid: String?
    public var ts: Date?
    public var whenread: String?
    public var isread: Bool?
    public var notificationtype: String?
    public var chatroomid: String?
    public var chatroomcustomid: String?
    public var commentconversationid: String?
    public var commentconversationcustomid: String?
    public var chateventid: String?
    public var commentid: String?
}

```

## 13.15 Delete User Notification

```

func deleteUserNotification(_ request: UserRequest.DeleteUserNotification,
    completionHandler: @escaping Completion<UserNotification>)

```

Deletes a User Notification

Immediately deletes a user notification. Unless your workflow specifically implements access to read notifications, you should delete notifications after they are consumed.

### Parameters

- **userid:** (required) The ID of the user marking the notification as read. This is used to ensure a user can't mark another user's notification as read.
- **notificationid:** (required) The unique ID of the notification being updated.

### Request Model: UserRequest.DeleteUserNotification

```

public class DeleteUserNotification {
    public var userid: String?
    public var notificationid: String?
}

```

### Response Model: UserNotification

```

open class UserNotification: Codable {
    public var kind: String?
    public var id: String?
    public var added: Date?
    public var userid: String?
    public var ts: Date?
    public var whenread: String?
    public var isread: Bool?
    public var notificationtype: String?
    public var chatroomid: String?

```

(continues on next page)

(continued from previous page)

```

public var chatroomcustomid: String?
public var commentconversationid: String?
public var commentconversationcustomid: String?
public var chateventid: String?
public var commentid: String?
}

```

## 13.16 Delete User Notification By ChatEventId

```

func deleteUserNotificationById(_ request: UserRequest.
↳DeleteUserNotificationByChatEventId, completionHandler: @escaping Completion
↳<UserNotification>)

```

Deletes a User Notification

Immediately deletes a user notification. Unless your workflow specifically implements access to read notifications, you should delete notifications after they are consumed.

### Parameters

- **userid:** (required) The ID of the user marking the notification as read. This is used to ensure a user can't mark another user's notification as read.
- **chateventid:** (required) The unique ID of the notification's chatEvent.

**Request Model:** `UserRequest.DeleteUserNotificationByChatEventId`

```

public class DeleteUserNotificationByChatEventId {
    public var userid: String?
    public var chateventid: String?
}

```

**Response Model:** `UserNotification`


```

open class UserNotification: Codable {
    public var kind: String?
    public var id: String?
    public var added: Date?
    public var userid: String?
    public var ts: Date?
    public var whenread: String?
    public var isread: Bool?
    public var notificationtype: String?
    public var chatroomid: String?
    public var chatroomcustomid: String?
    public var commentconversationid: String?
    public var commentconversationcustomid: String?
    public var chateventid: String?
    public var commentid: String?
}

```

## CHAT CLIENT

### 14.1 Create Room

```
func createRoom(_ request: ChatRequest.CreateRoom, completionHandler: @escaping  Completion<ChatRoom>)
```

Creates a new chat room

#### Parameters

- name: (required) The name of the room
- customid: (optional) A customid for the room. Can be unused, or a unique key.
- description: (optional) The description of the room
- moderation: (required) The type of moderation.
  - *pre* - marks the room as Premoderated
  - *post* - marks the room as Postmoderated
- enableactions: (optional) [true/false] Turns action commands on or off
- enableenterandexit: (optional) [true/false] Turn enter and exit events on or off. Disable for large rooms to reduce noise.
- enableprofanityfilter: (optional) [default=true / false] Enables profanity filtering.
- enableautoexpiresessions: (optional) [default=true / false] Enables automatically expiring idle sessions, which removes inactive users from the room.
- delaymessagesseconds: (optional) [default=0] Puts a delay on messages from when they are submitted until they show up in the chat. Used for throttling.
- maxreports: (optional) Default is 3. This is the maximum amount of user reported flags that can be applied to a message before it is sent to the moderation queue

**Warning** This method requires authentication

**Request Model:** `ChatRequest.CreateRoom`

```
public class CreateRoom {  
    public var name: String?  
    public var customid: String?  
    public var description: String?  
    public var moderation: String?  
    public var enableactions: Bool?
```

(continues on next page)

(continued from previous page)

```

public var enableenterandexit: Bool?
public var enableprofanityfilter: Bool?
public var roomisopen: Bool?
public var maxreports: Int? = 3
}

```

**Response Model: ChatRoom**

```

public var kind: String?
public var id: String?
public var appid: String?
public var ownerid: String?
public var name: String?
public var description: String?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var enableactions: Bool?
public var enableenterandexit: Bool?
public var open: Bool?
public var inroom: Int?
public var moderation: String?
public var maxreports: Int64?
public var enableprofanityfilter: Bool?
public var enableautoexpiresessions: Bool?
public var delaymessageseconds: Int64?
public var added: Date?
public var whenmodified: Date?
public var bouncedusers: [String] = []
public var reportedusers: [ReportedUser] = []
}

```

## 14.2 Get Room Details

```

func getRoomDetails(_ request: ChatRequest.GetRoomDetails, completionHandler: @escaping _
↳ Completion<ChatRoom>)

```

Get the details for a room

This will return all the settings for the room and the participant count but not the participant list

**Parameters**

- roomid: (required) Room id of a specific room againsts which you want to fetch the details

**Warning** This method requires authentication

**Request Model:** ChatRequest.GetRoomDetails



```
public class GetRoomDetails {
    public var roomid: String?
}
```

#### Response Model: ChatRoom

```
public var kind: String?
    public var id: String?
    public var appid: String?
    public var ownerid: String?
    public var name: String?
    public var description: String?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var enableactions: Bool?
    public var enableenterandexit: Bool?
    public var open: Bool?
    public var inroom: Int?
    public var moderation: String?
    public var maxreports: Int64?
    public var enableprofanityfilter: Bool?
    public var enableautoexpiresessions: Bool?
    public var delaymessageseconds: Int64?
    public var added: Date?
    public var whenmodified: Date?
    public var bouncedusers: [String] = []
    public var reportedusers: [ReportedUser] = []
}
```

## 14.3 Get Room Extended Details

```
func getRoomExtendedDetails(_ request: ChatRequest.GetRoomExtendedDetails,
    ↪completionHandler: @escaping Completion<ChatRoom>)
```

Get the details for a room

This method lets you specify a list of entity types to return. You can use it to get room details as well as statistics and other data associated with a room that is not part of the room entity.

You must specify one or more roomid values or customid values. You may optionally provide both roomid and customid values. You may not request more than 20 rooms at once total. You must specify at least one entity type.

In the future, each entity requested will count towards your API usage quota, so don't request data you will not be using.

The response will be a list of RoomExtendedDetails objects. They contain properties such as room, mostrecentmessage, and inroom. These properties will be null if their entity type is not specified

#### Parameters

- roomid: (required) Room id of a specific room againsts which you want to fetch the details

- customid: (optional) A list of room customIDs.
- entity: (required) Specify one or more ENTITY TYPES to include in the response. Use one or more of the types below.
  - room: This returns the room entity.
  - numparticipants: This returns number of active participants / room subscribers.
  - lastmessagetime: This returns the time stamp for the most recent event that is a visible displayable message (speech, quote, threaded reply or announcement).

**Warning** This method requires authentication

**Request Model: ChatRequest.GetRoomExtendedDetails**

```
public class GetRoomExtendedDetails {  
    public var roomid: String?  
    public var customid: String?  
    public var entity: [RoomEntityType]?  
}
```

**Response Model: ChatRoom**

```
public var kind: String?  
public var id: String?  
public var appid: String?  
public var ownerid: String?  
public var name: String?  
public var description: String?  
public var customtype: String?  
public var customid: String?  
public var custompayload: String?  
public var customtags: [String]?  
public var customfield1: String?  
public var customfield2: String?  
public var enableactions: Bool?  
public var enableenterandexit: Bool?  
public var open: Bool?  
public var inroom: Int?  
public var moderation: String?  
public var maxreports: Int64?  
public var enableprofanityfilter: Bool?  
public var enableautoexpiresessions: Bool?  
public var delaymessageseconds: Int64?  
public var added: Date?  
public var whenmodified: Date?  
public var bouncedusers: [String] = []  
public var reportedusers: [ReportedUser] = []  
}
```

## 14.4 Get Room Details By Custom ID

```
func getRoomDetailsByCustomId(_ request: ChatRequest.GetRoomDetailsByCustomId,
    ↪completionHandler: @escaping Completion<ChatRoom>)
```

Get the details for a room

This will return all the settings for the room and the participant count but not the participant list

### Parameters

- customid: Custom Id of a specific room againsts which you want to fetch the details.

**Warning** This method requires authentication

**Request Model:** ChatRequest.GetRoomDetails

```
public class GetRoomDetailsByCustomId {
    public var customid: String?
}
```

**Response Model:** ChatRoom

```
public var kind: String?
public var id: String?
public var appid: String?
public var ownerid: String?
public var name: String?
public var description: String?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var enableactions: Bool?
public var enableenterandexit: Bool?
public var open: Bool?
public var inroom: Int?
public var moderation: String?
public var maxreports: Int64?
public var enableprofanityfilter: Bool?
public var enableautoexpiresessions: Bool?
public var delaymessageseconds: Int64?
public var added: Date?
public var whenmodified: Date?
public var bouncedusers: [String] = []
public var reportedusers: [ReportedUser] = []
}
```

## 14.5 Delete Room

```
func deleteRoom(_ request: ChatRequest.DeleteRoom, completionHandler: @escaping _  
↳Completion<DeleteChatRoomResponse>)
```

Permanently deletes a chat room

This cannot be reversed. This command permanently deletes the chat room and all events in it.

### Parameters

- roomid: (required) that you want to delete

**Warning** This method requires authentication

**Request Model:** ChatRequest.DeleteRoom

```
public class DeleteRoom {  
    public var roomid: String?  
}
```

**Response Model:** DeleteChatRoomResponse

```
public struct DeleteChatRoomResponse: Codable {  
    public var kind: String?  
    public var deletedEventsCount: Int64?  
    public var room: ChatRoom?  
}
```

## 14.6 Update Room

```
func updateRoom(_ request: ChatRequest.UpdateRoom, completionHandler: @escaping _  
↳Completion<ChatRoom>)
```

Updates an existing room

### Parameters

- roomid: (required) The ID of the existing room.
- userid: (optional) The owner of the room.
- name: (optional) The name of the room.
- description: (optional) The description of the room.
- moderation: (optional) [premoderation/postmoderation] Defaults to post-moderation.
- enableactions: (optional) [true/false] Turns action commands on or off.
- enableenterandexit: (optional) [true/false] Turn enter and exit events on or off. Disable for large rooms to reduce noise.
- enableprofanityfilter: (optional) [default=true / false] Enables profanity filtering.
- enableautoexpiresessions: (optional) [defaulttrue / false] Enables automatically expiring idle sessions, which removes inactive users from the room.

- `delaymessageseconds`: (optional) [default=0] Puts a delay on messages from when they are submitted until they show up in the chat. Used for throttling
- `roomisopen`: (optional) [true/false] If false, users cannot perform any commands in the room, chat is suspended.
- `throttle`: (optional) [default=0] This is the number of seconds to delay new incoming messages so that the chat room doesn't scroll messages too fast

**Warning** This method requires authentication

**Request Model: ChatRequest.UpdateRoom**

```
public class UpdateRoom {
    public var roomid: String?
    public var name: String?
    public var description: String?
    public var customid: String?
    public var moderation: String?
    public var enableactions: Bool?
    public var enableenterandexit: Bool?
    public var enableprofanityfilter: Bool?
    public var delaymessageseconds: Int?
    public var roomisopen: Bool?
    public var throttle: Int?
    public var userid: String?
}
```

**Response Model: ChatRoom**

```
public var kind: String?
    public var id: String?
    public var appid: String?
    public var ownerid: String?
    public var name: String?
    public var description: String?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var enableactions: Bool?
    public var enableenterandexit: Bool?
    public var open: Bool?
    public var inroom: Int?
    public var moderation: String?
    public var maxreports: Int64?
    public var enableprofanityfilter: Bool?
    public var enableautoexpiresessions: Bool?
    public var delaymessageseconds: Int64?
    public var added: Date?
    public var whenmodified: Date?
    public var bouncedusers: [String] = []
    public var reportedusers: [ReportedUser] = []
}
```

## 14.7 Update and Close Room

```
func updateCloseRoom(_ request: ChatRequest.UpdateRoomCloseARoom, completionHandler: @escaping Completion<ChatRoom>)
```

Updates an existing room

### Parameters

- roomid: (required) The ID of the existing room
- name: (optional) The name of the room
- description: (optional) The description of the room
- moderation: (optional) [premoderation/postmoderation] Defaults to post-moderation.
- enableactions: (optional) [true/false] Turns action commands on or off
- enableenterandexit: (optional) [true/false] Turn enter and exit events on or off. Disable for large rooms to reduce noise.
- enableprofanityfilter: (optional) [default=true / false] Enables profanity filtering.
- delaymessagesseconds: (optional) [default=0] Puts a delay on messages from when they are submitted until they show up in the chat. Used for throttling.
- roomisopen: (optional) [true/false] If false, users cannot perform any commands in the room, chat is suspended.

**Warning** This method requires authentication

**Request Model:** ChatRequest.UpdateRoomCloseARoom

```
public class UpdateRoomCloseARoom {
    public var roomid: String?
    public var name: String?
    public var description: String?
    public var moderation: String?
    public var enableactions: Bool?
    public var enableenterandexit: Bool?
    public var enableprofanityfilter: Bool?
    public var delaymessagesseconds: Int?
    public var roomisopen: Bool? = false
    public var userid: String?
}
```

**Response Model:** ChatRoom

```
public var kind: String?
public var id: String?
public var appid: String?
public var ownerid: String?
public var name: String?
public var description: String?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
```

(continues on next page)

(continued from previous page)

```

public var customfield2: String?
public var enableactions: Bool?
public var enableenterandexit: Bool?
public var open: Bool?
public var inroom: Int?
public var moderation: String?
public var maxreports: Int64?
public var enableprofanityfilter: Bool?
public var enableautoexpiresessions: Bool?
public var delaymessageseconds: Int64?
public var added: Date?
public var whenmodified: Date?
public var bouncedusers: [String] = []
public var reportedusers: [ReportedUser] = []
}

```

## 14.8 List Rooms

```

func listRooms(_ request: ChatRequest.ListRooms, completionHandler: @escaping Completion
    <<ListRoomsResponse>)

```

List all the available public chat rooms

Rooms can be public or private. This method lists all public rooms that everyone can see.

### Parameters

- cursor: (optional) The first time you call list rooms, omit this property to start from the beginning. Call the method again passing in the value returned in the cursor field of the response to get the next page of results. If there are more results available, more will be true.
- limit: (optional) Specify the number of items to return. Default is 200

**Warning** This method requires authentication

**Request Model:** ChatRequest.ListRooms

```

public class ListRooms {
    public var cursor: String?
    public var limit: Int = 200
}

```

Response Model: ListRoomsResponse

```

public struct ListRoomsResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var rooms: [ChatRoom]
}

```

## 14.9 List Room Participants

```
func listRoomParticipants(_ request: ChatRequest.ListRoomParticipants,
↳ completionHandler: @escaping Completion<ListChatRoomParticipantsResponse>)
```

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 invocation 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.

### Parameters

- roomid: (required) room id that you want to list the participants
- cursor: (optional) you can pass that value to the next invocation to get the next page of results
- limit: (optional) default is 200

**Warning** This method requires authentication

**Request Model:** ChatRequest.ListRoomParticipants

```
public class ListRoomParticipants {
    public var roomid: String?
    public var cursor: String? = ""
    public var limit: Int? = 200
}
```

**Response Model:** ListChatRoomParticipantsResponse

```
public struct ListChatRoomParticipantsResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var participants: [ChatRoomParticipant]
}
```

## 14.10 List User Subscribed Rooms

```
func listUserSubscribedRooms(_ request: ChatRequest.ListUserSubscribedRooms,
↳ completionHandler: @escaping Completion<ListUserSubscribedRoomResponse>)
```

List the rooms the user is subscribed to.

Use this method to cursor through all the rooms the user is subscribed to. This will include all rooms. If you want to build a private messaging experience, you can put custom tags on the rooms to separate out which are for private messenger and which are public group rooms.

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 invocation 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.

#### Parameters

- userid: (required)
- cursor: (optional) you can pass that value to the next invocation to get the next page of results
- limit: (optional) default is 200

**Warning** This method requires authentication

#### Request Model: ChatRequest.ListUserSubscribedRooms

```
public class ListUserSubscribedRooms {
    public var userid: String?
    public var cursor: String? = ""
    public var limit: Int? = 200
}
```

#### Response Model: ListUserSubscribedRoomsResponse

```
public struct ListUserSubscribedRoomsResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var subscriptions: [ChatSubscription] = []
}
```

## 14.11 List Event History

```
func listEventHistory(_ request: ChatRequest.ListEventHistory, completionHandler: @escaping Completion<ListEventsResponse>)
```

- 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

#### Parameters

- roomid: (required) Room id where you want event history to be listed
- limit: (optional) default is 100, maximum 2000
- cursor: (optional) If not provided, the most recent events will be returned. To get older events, call this method again using the cursor string returned from the previous call.

#### Request Model: ChatRequest.ListEventHistory

```
public class ListEventHistory {
    public var roomid: String?
    public var cursor: String? = ""
}
```

(continues on next page)

(continued from previous page)

```

    public var limit: Int? = 100
}

```

**Response Model: ListEventsResponse**

```

public struct ListEventsResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var events: [Event]
}

```

## 14.12 List Previous Events

```

func listPreviousEvents(_ request: ChatRequest.ListPreviousEvents, completionHandler: @escaping Completion<ListEventsResponse>)

```

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

**Parameters**

- roomid: (required) Room id where you want previous events to be listed
- limit: (optional) default is 100, maximum 500
- cursor: (optional) If not provided, the most recent events will be returned. To get older events, call this method again using the cursor string returned from the previous call.

**Request Model: ChatRequest.ListPreviousEvents**

```

public class ListPreviousEvents {
    public var roomid: String?
    public var cursor: String?
    public var limit: Int? = 100
}

```

**Response Model: ListEventsResponse**

```

public struct ListEventsResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
}

```

(continues on next page)

(continued from previous page)

```
public var events: [Event]
}
```

## 14.13 List Event By Type

```
func listEventByType(_ request: ChatRequest.ListEventByType, completionHandler: _
↳ @escaping Completion<ListEventsResponse>)
```

- This method enables you to retrieve a small list of recent events by type. This is useful for things like fetching a list of recent announcements or custom event types without the need to scroll through the entire chat history.
- This method returns a list of events sorted from newest to oldest.
- This method returns only active events.

### Parameters

- roomid: (required) Room id where you want previous events to be listed
- limit: (optional) default is 10, maximum 100
- cursor: (optional) If not provided, the most recent events will be returned. To get older events, call this method again using the cursor string returned from the previous call.
- eventtype: (required) Specify the chat event type you are filtering for. If you want to filter for a custom event type, specify 'custom' and then provide a value for the *\*customtype* parameter
- customtype: (optional) If you want to filter by custom type you must first specify 'custom' for the eventtype field. This will enable you to filter to find events of a custom type

### Request Model: ChatRequest.ListEventByType

```
public class ListPreviousEvents {
    public var roomid: String?
    public var eventtype: EventType?
    public var cursor: String?
    public var limit: Int? = 10
}
```

### Response Model: ListEventsResponse

```
public struct ListEventsResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var events: [Event]
}
```

## 14.14 List Event By Timestamp

```
func listEventByTimestamp(_ request: ChatRequest.ListEventByTimestamp, completionHandler: @escaping Completion<ListEventsResponse>)
```

- This method enables you to retrieve an event using a timestamp.
- You can optionally retrieve a small number of displayable events before and after the message at the requested timestamp.
- This method returns a list of events sorted from oldest to newest.
- This method returns only active events.
- The timestamp is a high resolution timestamp accurate to the thousandth of a second. It is possible, but very unlikely, for two messages to have the same timestamp.
- The method returns “timestampolder”. This can be passed as the timestamp value when calling functions like this which accept a timestamp to retrieve data.
- The method returns “timestampnewer”. This can be passed as the timestamp value when calling this function again.
- The method returns “cursorolder”. This can be passed as the cursor to methods that accept an events-sorted-by-time cursor.
- The method returns “cursornewer”. This can be passed as the cursor to methods that accept an events-sorted-by-time cursor.

### Limitation

If you pass in 0 for limitolder you won't get any older events than your timestamp and hasmoreolder will always be false because the API will not query for older events. If you pass in 0 for limitnewer you won't get any newer events than your timestamp and hasmorenewer will always be false because the API will not query for newer events

### Parameters

- roomid: (required) Room id where you want previous events to be listed
- ts: (required) If not provided, the most recent events will be returned. To get older events, call this method again using the cursor string returned from the previous call
- limitolder: (optional) Defaults to 0, maximum 100.
- limitnewer : (optional) Defaults to 0, maximum 100

### Request Model: ChatRequest.ListEventByType

```
public class ListPreviousEvents {
    public var roomid: String?
    public var timestamp: Int?
    public var limitolder: Int? = 0
    public var limitnewer: Int? = 0
}
```

### Response Model: ListEventByTimestampResponse

```
public struct ListEventByTimestampResponse: Codable {
    public var kind: String?
    public var cursorolder: String?
    public var cursornewer: String?
```

(continues on next page)

(continued from previous page)

```

public var timestampolder: Int?
public var timestampnewer: Int?
public var hasmoreolder: Bool?
public var hasmorenewer: Bool?
public var itemcount: Int64?
public var events: [Event]
}

```

## 14.15 Join Room

```

func joinRoom(_ request: ChatRequest.JoinRoom, completionHandler: @escaping Completion
    <>JoinChatRoomResponse>)

```

### Join A Room

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

- Create Room
- Create User
- Join Room (user gets permission to access events data from the room)
- 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

### DATA PARAMETERS

Provide a unique user ID string and chat handle string. If this is the first time the user ID has been used a new user record will be created for the user. Whenever the user creates an event in the room by doing an action like saying something, the user information will be returned.

You can optionally also provide a URL to an image and a URL to a profile.

If you provide user information and the user already exists in the database, the user will be updated with the new information.

The user will be added to the list of participants in the room and the room participant count will increase.

The user will be removed from the room automatically after some time if the user doesn't perform any operations.

Users can only execute commands in the room if they have joined the room.

When a logged in user joins a room an entrance event is generated in the room.

When a logged in user leaves a room, an exit event is generated in the room

**Creating A New User:** You have the option to create or update an existing user during join.

### Parameters

- limit: (optional) Defaults to 50. This limits the number of previous messages returned when joining the room.
- userid: (required) If the userid is new then the user will be created. If the userid is already in use in the database then the user will be updated.
- handle: (Optional) A unique string representing the user that is easy for other users to type.
  - Example @GeorgeWashington could be the handle but Display Name could be "Wooden Teef For The Win".

- If you are creating a user and you don't specify a handle, the system will generate one for you (using Display Name as basis if you provide that).
  - If you request a handle and it's already in use a new handle will be generated for you by adding a number from 1-99 and returned.
  - If the handle can't be generated because all the options 1-99 on the end of it are taken then the request will be rejected with BadRequest status code.
  - Only these characters may be used: “*abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\_*”
- displayname: (optional) This is the desired name to display, typically the real name of the person.
  - pictureurl: (optional) The URL to the picture for this user.
  - profileurl: (optional) The profileurl for this user.

**Warning** This method requires authentication

#### Request Model: ChatRequest.JoinRoom

```
public class JoinRoom {
    public var roomid: String?
    public var userid: String?
    public var handle: String?
    public var displayname: String?
    public var pictureurl: URL?
    public var profileurl: URL?
    public var limit: Int? = 50
}
```

#### Response Model: JoinChatRoomResponse

```
public struct JoinChatRoomResponse: Codable {
    public var kind: String?
    public var user: User?
    public var room: ChatRoom?
    public var eventscursor: GetUpdatesResponse?
}
```

## 14.16 Join Room by CustomId

```
func joinRoomByCustomId(_ request: ChatRequest.JoinRoomByCustomId, completionHandler: @escaping Completion<JoinChatRoomResponse>)
```

### 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.

When you attempt to join the room, if the userid you provide does not exist then a user will be created for you automatically.

If you provide a Display Name and you do not provide a handle then the display name will automatically be used to generate a handle for you. If you do not provide a display name or a handle then a 16 character handle will be automatically generated for you.

## DATA PARAMETERS

Provide a unique user ID string and chat handle string. If this is the first time the user ID has been used a new user record will be created for the user. Whenever the user creates an event in the room by doing an action like saying something, the user information will be returned.

You can optionally also provide a URL to an image and a URL to a profile.

If you provide user information and the user already exists in the database, the user will be updated with the new information.

The user will be added to the list of participants in the room and the room participant count will increase.

The user will be removed from the room automatically after some time if the user doesn't perform any operations.

Users can only execute commands in the room if they have joined the room.

When a logged in user joins a room an entrance event is generated in the room.

When a logged in user leaves a room, an exit event is generated in the room.

**Creating A New User:** You have the option to create or update an existing user during join.

### Parameters

- limit: (optional) Defaults to 50. This limits the number of previous messages returned when joining the room.
- userid: (required). If the userid is new then the user will be created. If the userid is already in use in the database then the user will be updated.
- handle: (Optional) A unique string representing the user that is easy for other users to type.
  - Example @GeorgeWashington could be the handle but Display Name could be "Wooden Teef For The Win".
  - If you are creating a user and you don't specify a handle, the system will generate one for you (using Display Name as basis if you provide that).
  - If you request a handle and it's already in use a new handle will be generated for you by adding a number from 1-99 and returned.
  - If the handle can't be generated because all the options 1-99 on the end of it are taken then the request will be rejected with BadRequest status code.
  - Only these characters may be used: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\_"
- displayname: (optional) This is the desired name to display, typically the real name of the person.
- pictureurl: (optional) The URL to the picture for this user.
- profileurl: (optional) The profileurl for this user.

**Warning** This method requires authentication

**Request Model:** ChatRequest.JoinRoomByCustomId

```
public class JoinRoomByCustomId {  
    public var customid: String?  
    public var userid: String?  
    public var handle: String?  
    public var displayname: String?  
    public var pictureurl: URL?  
    public var profileurl: URL?  
    public var limit: Int? = 50  
}
```

**Response Model:** JoinChatRoomResponse

```
public struct JoinChatRoomResponse: Codable {  
    public var kind: String?  
    public var user: User?  
    public var room: ChatRoom?  
    public var eventscursor: GetUpdatesResponse?  
}
```

## 14.17 Exit Room

```
func exitRoom(_ request: ChatRequest.ExitRoom, completionHandler: @escaping Completion  
↳ <ExitChatRoomResponse>)
```

Exit a Room

This method should be called to remove a user from a room. This will cause an EXIT event to be broadcast in the room and this user will no longer show up in the list of attendees in the room.

### Parameters

- roomid: (required) Room id that you want to exit
- userid: (required) user id specific to App

**Warning** This method requires authentication

**Request Model:** ChatRequest.ExitRoom

```
public class ExitRoom {  
    public var roomid: String?  
    public var userid: String?  
}
```

**Response Model:** ExitChatRoomResponse

```
public struct ExitChatRoomResponse: Codable {  
    public var kind: String?  
}
```



## 14.18 Get Updates

```
func getUpdates(_ request: ChatRequest.GetUpdates, completionHandler: @escaping _
↳Completion<GetUpdatesResponse>)
```

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.

Each event in the stream has a KIND property. Inspect the property to determine if it is a;

- enter event: A user has joined the room.
- exit event: A user has exited chat.
- message: A user has communicated a message.
- reply: A user sent a message in response to another user.
- reaction: A user has reacted to a message posted by another user.
- action: A user is performing an ACTION (emote) alone or with another user.

### Enter and Exit Events

Enter and Exit events may not be sent if the room is expected to have a very large number of users.

### Parameters

- roomid: (required) Room id that you want to update
- cursor: (optional) Used in cursoring through the list. Gets the next batch of users. Read 'nextCur' property of result set and pass as cursor value.
- limit: (optional) Number of events to return. Default is 100, maximum is 500

**Warning** This method requires authentication

### Request Model: ChatRequest.GetUpdates

```
public class GetUpdates {
    public var roomid: String?
    public var cursor: String?
    public var limit: Int = 100
}
```

### Response Model: GetUpdatesResponse

```
public struct GetUpdatesResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var room: ChatRoom?
    public var events: [Event]
}
```

## 14.19 Get More Updates

```
func getMoreUpdates(_ request: ChatRequest.GetMoreUpdates, completionHandler: @escaping _
↳ Completion<GetUpdatesResponse>)
```

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.

Each event in the stream has a KIND property. Inspect the property to determine if it is a;

- enter event: A user has joined the room.
- exit event: A user has exited chat.
- message: A user has communicated a message.
- reply: A user sent a message in response to another user.
- reaction: A user has reacted to a message posted by another user.
- action: A user is performing an ACTION (emote) alone or with another user.

### Enter and Exit Events

Enter and Exit events may not be sent if the room is expected to have a very large number of users.

### Parameters

- roomid: (required) Room id that you want to update
- cursor: (optional) Used in cursoring through the list. Gets the next batch of users. Read 'nextCur' property of result set and pass as cursor value.
- limit: (optional) Number of events to return. Default is 100, maximum is 500

**Warning** This method requires authentication

### Request Model: ChatRequest.GetUpdates

```
public class GetMoreUpdates {
    public var roomid: String?
    public var cursor: String?
    public var limit: Int = 100
}
```

### Response Model: GetUpdatesResponse

```
public struct GetUpdatesResponse: Codable {
    public var kind: String?
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var room: ChatRoom?
    public var events: [Event]
}
```

## 14.20 Execute Command

```
func executeChatCommand(_ request: ChatRequest.ExecuteChatCommand, completionHandler: @escaping Completion<ExecuteChatCommandResponse>) throws
```

Executes 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)

### API UPDATES

- replyto: This is deprecated. For replies use Quoted Reply or Threaded Reply. For most use cases, Quoted Reply is the recommended approach.

### SENDING A MESSAGE

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

*example* These commands both do the same thing, which is send the message "Hello World" to the room. SAY Hello, World

### ACTION COMMANDS

- Action commands start with the / character

*example*

*/dance nicole* User sees: *You dance with Nicole* Nicole sees: *(user's handle) dances with you* Everyone else sees: *(user's handle) 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.

### ADMIN COMMANDS

- These commands start with the \* character

*example* - ban : This bans the user from the entire chat experience (all rooms).

- restore : This restores the user to the chat experience (all rooms).
- purge : This deletes all messages from the specified user.
- deleteallevents : This deletes all messages in this room.

### Parameters

- command: (required) The command to execute. See examples above.
- userid: (required) The userid of user who is executing the command. The user must have joined the room first.
- eventtype: (optional, default = speech) By default, the API will determine the type of event by processing your command. However you can send custom commands.
- custom : This indicates you will be using a custom event type.
- announcement : This indicates the event is of type announcement.
- ad : Use this event type to push an advertisement. Use the CustomPayload property to specify parameters for your add.

- **customtype:** (optional) A string having meaning to your app that represents a custom type of event defined by you. You must specify “custom” as the eventtype to use this. If you don’t, the event type will be forced to custom anyway.
- **custompayload:** (optional) A string (XML or JSON usually) representing custom data for your application to use.
- **replyto:** (optional) Use this field to provide the EventID of an event you want to reply to. Replies have a different event type and contain a copy of the original event.
- **moderation:** (optional) Use this field to override the moderation state of the chat event. Use this when you have already inspected the content. Use one of the values below.
- **approved :** The content has already been approved by a moderator and it should not be sent to the moderation queue if users report it since the decision was already made to approve it.
- **prescreened :** The content was prescreened, but not approved. This means it can still be flagged for moderation queue by the users. This state allows a data analyst to distinguish between content that was approved by a moderator and content that went through a filtering process but wasn’t explicitly approved or rejected.
- **rejected :** The content has been rejected by a moderator and it should not be broadcast into the chat stream, but it should be saved to the chat room history for future analysis or audit trail purposes.

## RESPONSE CODES

200 | OK : Sweet, sweet success.

400 | BadRequest : Something is wrong with your request. View response message and errors list for details.

403 | Forbidden : The userid issuing the request is banned from chatting in this room (or is banned globally).

405 | MethodBlocked : The method was blocked because it contained profanity and filtermode was set to ‘block’.

409 | Conflict : The customid of your event is already in use.

412 | PreconditionFailed : User must JOIN the room before executing a chat command.

### Request Model: ChatRequest.ExecuteChatCommand

```
public class ExecuteChatCommand {  
    public var roomid: String?  
    public var command: String?  
    public var userid: String?  
    public var moderation: String?  
    public var eventtype: EventType?  
    public var customtype: String?  
    public var customid: String?  
    public var custompayload: String?  
}
```

### Response Model: ExecuteChatCommandResponse

```
public struct ExecuteChatCommandResponse: Codable {  
    public var kind: String?  
    public var op: String?  
    public var room: ChatRoom?  
    public var speech: Event?  
    public var action: Event?  
}
```

## 14.21 Send Quoted Reply

```
func sendQuotedReply(_ request: ChatRequest.SendQuotedReply, completionHandler: @escaping Completion<Event>) throws
```

Quotes an existing message and republishes it with a new message

This method is provided to support a chat experience where a person wants to reply to another person, and the reply is inline with the rest of chat, but contains a copy of all or part of the original message you are replying to. You can see this behavior in WhatsApp and iMessage. This way, when viewing the reply, the user doesn't need to scroll up searching conversation history for the context (the parent the reply is addressing).

### Parameters

- **eventid:** (required) The ID of the event you are quoting
- **userid:** (required) The userid of the user who is publishing the quoted reply.
- **body:** (required) The contents of the reply for the quoted reply. Cannot be empty.
- **customid:** (optional) Assigns a custom ID to the quoted reply event.
- **custompayload:** (optional) Attach a custom payload string to the quoted reply such as JSON or XML.
- **customfield1:** (optional) Use this field however you wish.
- **customfield2:** (optional) Use this field however you wish.
- **customtags:** (optional) An array of strings, use this field however you wish.

### Request Model: ChatRequest.SendQuotedReply

```
public class SendQuotedReply {
    public var roomid: String?
    public var eventId: String?
    public var userid: String?
    public var body: String?
    public var customid: String?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?
    public var customtags: String?
}
```

### Response Model: Event

```
open class Event: Codable, Equatable {
    public var kind: String?
    public var id: String?
    public var roomid: String?
    public var body: String?
    public var originalbody: String?
    public var added: Date?
    public var modified: Date?
    public var ts: Date?
    public var eventtype: EventType?
    public var userid: String?
    public var user: User?
```

(continues on next page)

(continued from previous page)

```

public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var replyto: Event?
public var parentid: String?
public var edited: Bool?
public var editedby moderator: Bool?
public var censored: Bool?
public var deleted: Bool?
public var active: Bool?
public var shadowban: Bool?
public var likecount: Int64?
public var replycount: Int64?
public var reactions: [ChatEventReaction]
public var moderation: String?
public var reports: [ChatEventReport]
}

```

## 14.22 Send Threaded Reply

```

func sendThreadedReply(_ request: ChatRequest.SendThreadedReply, completionHandler: (
    @escaping Completion<Event>) throws

```

Creates a threaded reply to another message event

The purpose of this method is to enable support of a sub-chat within the chat room. You can use it to split off the conversation into a nested conversation. You can build a tree structure of chat messages and replies, but it is recommended not to build experiences deeper than parent and child conversation level or it becomes complex for the users to follow.

Replies do not support admin or action commands

### Parameters

- eventid: (required) The ID of the event you are quoting
- userid: (required) The userid of the user who is publishing the quoted reply.
- body: (required) The contents of the reply for the quoted reply. Cannot be empty.
- customid: (optional) Assigns a custom ID to the quoted reply event.
- custompayload: (optional) Attach a custom payload string to the quoted reply such as JSON or XML.
- customfield1: (optional) Use this field however you wish.
- customfield2: (optional) Use this field however you wish.
- customtags: (optional) An array of strings, use this field however you wish.

**Request Model:** `ChatRequest.SendThreadedReply`

```

public class SendThreadedReply {
    public var roomid: String?
    public var eventid: String?
    public var userid: String?
    public var body: String?
    public var customid: String?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?
    public var customtags: String?
}

```

#### Response Model: Event

```

open class Event: Codable, Equatable {
    public var kind: String?
    public var id: String?
    public var roomid: String?
    public var body: String?
    public var originalbody: String?
    public var added: Date?
    public var modified: Date?
    public var ts: Date?
    public var eventtype: EventType?
    public var userid: String?
    public var user: User?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var replyto: Event?
    public var parentid: String?
    public var edited: Bool?
    public var editedbymoderator: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var active: Bool?
    public var shadowban: Bool?
    public var likecount: Int64?
    public var replycount: Int64?
    public var reactions: [ChatEventReaction]
    public var moderation: String?
    public var reports: [ChatEventReport]
}

```

## 14.23 List Messages By User

```
func listMessagesByUser(_ request: ChatRequest.ListMessagesByUser, completionHandler: @escaping Completion<ListMessagesByUser>)
```

Gets a list of users messages

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.

### Parameters

- roomid: (required) Room id, in which you want to fetch messages
- userid: (required) user id, against which you want to fetch messages
- cursor: (optional) Used in cursoring through the list. Gets the next batch of users. Read 'nextCur' property of result set and pass as cursor value.
- limit: (optional) default 200

**Warning** This method requires authentication

**Request Model:** ChatRequest.ListMessagesByUser

```
public class ListMessagesByUser {
    public var cursor: String?
    public var limit: Int? = 200
    public var userId: String?
    public var roomId: String?
}
```

**Response Model:** ListMessagesByUser

```
public struct ListMessagesByUser: Codable {
    public var kind: String?
    public var cursor: String?
    public var events: [Event]
}
```

## 14.24 Purge Message

```
func purgeMessage(_ request: ChatRequest.PurgeUserMessages, completionHandler: @escaping Completion<ExecuteChatCommandResponse>)
```

Executes a command in a chat room to purge all messages for a user

This does not DELETE the message. It flags the message as moderator removed.

### Parameters

- roomid: (required)
- userid: (required) the id of the owner of the messages
- handle: (required) the handle of the owner of the messages



- password: (required) a valid admin password

**Warning** This method requires authentication

**Request Model:** ChatRequest.PurgeUserMessages

```
public class PurgeUserMessages {
    public var roomid: String?
    public var userid: String?
    public var handle: String?
    public var password: String?
    private var command: String!
}
```

**Response Model:** ExecuteChatCommandResponse

```
public struct ExecuteChatCommandResponse: Codable {
    public var kind: String?
    public var op: String?
    public var room: ChatRoom?
    public var speech: Event?
    public var action: Event?
}
```

## 14.25 Flag Event As Locally Deleted

```
func flagEventLogicallyDeleted(_ request: ChatRequest.FlagEventLogicallyDeleted,
    ↪completionHandler: @escaping Completion<DeleteEventResponse>)
```

Set Deleted (LOGICAL DELETE)

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).

### Parameters

- roomid: (required) The ID of the room containing the event
- eventid: (required) The unique ID of the chat event to delete. The user posting the delete request must be the owner of the event or have moderator permission
- userid: (required) This is the application specific user ID of the user deleting the comment. Must be the owner of the message event or authorized moderator.
- deleted: (required) Set to true or false to flag the comment as deleted. If a comment is deleted, then it will have the deleted field set to true, in which case the contents of the event message should not be shown and the body

of the message will not be returned by the API by default. If a previously deleted message is undeleted, the flag for deleted is set to false and the original comment body is returned

- `permanentifnoreplies`: (optional) If this optional parameter is set to “true”, then if this event has no replies it will be permanently deleted instead of logically deleted. If a permanent delete is performed, the result will include the field “`permanentdelete=true`”

If you want to mark a comment as deleted, and replies are still visible, use “true” for the logical delete value. If you want to permanently delete the message and all of its replies, pass false

**Request Model: `ChatRequest.FlagEventLogicallyDeleted`**

```
public class FlagEventLogicallyDeleted {
    public var roomid: String?
    public var eventid: String?
    public var userid: String?
    public var deleted: Bool?
    public var permanentifnoreplies: Bool?
}
```

**Response Model: `ListMessagesByUser`**

```
public struct ListMessagesByUser: Codable {
    public var kind: String?
    public var cursor: String?
    public var events: [Event]
}
```

## 14.26 Permanently Delete Event

```
func permanentlyDeleteEvent(_ request: ChatRequest.PermanentlyDeleteEvent,
    ↪completionHandler: @escaping Completion<DeleteEventResponse>)
```

Deletes an event from the room.

This does not DELETE the message. It flags the message as moderator removed.

### Parameters

- `roomid`: (required) the room id in which you want to remove the message
- `eventId`: (required) the message you want to remove.
- `userid`: (optional) If provided, a check will be made to enforce this `userid` (the one deleting the event) is the owner of the event or has elevated permissions. If null, it assumes your business service made the determination to delete the event. If it is not provided this authorization check is bypassed.

**Warning** This method requires authentication

**Request Model: `ChatRequest.PermanentlyDeleteEvent`**

```
public class PermanentlyDeleteEvent {
    public var roomid: String?
    public var eventid: String?
    public var userid: String?
}
```

**Response Model: DeleteEventResponse**

```
public struct DeleteEventResponse: Codable {
    public var kind: String?
    public var permanentdelete: Bool?
    public var event: Event?
}
```

## 14.27 Delete All Events

```
func deleteAllEvents(_ request: ChatRequest.DeleteAllEvents, completionHandler: @escaping Completion<ExecuteChatCommandResponse>)
```

Deletes all the events in a room.

**Parameters**

- roomid: (required)
- userid: (required) the id of the owner of the messages
- password: (required) a valid admin password

**Request Model: ChatRequest.DeleteAllEvents**

```
public class DeleteAllEvents {
    public var roomid: String?
    private var command: String?
    public var password: String?
    public var userid: String?
}
```

**Response Model: ExecuteChatCommandResponse**

```
public struct ExecuteChatCommandResponse: Codable {
    public var kind: String?
    public var op: String?
    public var room: ChatRoom?
    public var speech: Event?
    public var action: Event?
}
```

## 14.28 List Messages of User

```
func listMessagesByUser(_ request: ChatRequest.ListMessagesByUser, completionHandler: @escaping Completion<ListMessagesByUserResponse>)
```

Gets a list of users messages

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.

**Parameters**

- roomid: (required) Room id, in which you want to fetch messages
- userid: (required) user id, against which you want to fetch messages
- cursor: (optional) Used in cursoring through the list. Gets the next batch of users. Read 'nextCur' property of result set and pass as cursor value.
- limit: (optional) default 200

**Warning** This method requires authentication


**Request Model:** ChatRequest.ListMessagesByUser

```
public class ListMessagesByUser {  
    public var cursor: String?  
    public var limit: String? = defaultLimit  
    public var userId: String?  
    public var roomid: String?  
}
```

**Response Model:** ListMessagesByUserResponse

```
public struct ListMessagesByUserResponse: Codable {  
    public var kind: String?  
    public var cursor: String?  
    public var events: [Event]  
}
```

## 14.29 Report A Message

```
func reportMessage(_ request: ChatRequest.ReportMessage, completionHandler: @escaping  Completion<Event>)
```

Reports a message to the moderation team

A reported message is temporarily removed from the chat event stream until it is evaluated by a moderator.

### Parameters

- roomid: the id of the room in which you want to report the event
- eventid: the id of the event that you want to report.
- userid: (required) user id specific to app
- reporttype: (required) [defaults="abuse"] e.g. abuse

**Warning** This method requires authentication.

**Request Model:** ChatRequest.ReportMessage

```
public class ReportMessage {  
    public var roomid: String?  
    public var eventid: String?  
    public var userid: String?  
    public var reporttype = "abuse"  
}
```

**Response Model:** Event

```

open class Event: Codable, Equatable {
    public var kind: String?
    public var id: String?
    public var roomid: String?
    public var body: String?
    public var originalbody: String?
    public var added: Date?
    public var modified: Date?
    public var ts: Date?
    public var eventtype: EventType?
    public var userid: String?
    public var user: User?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var replyto: Event?
    public var parentid: String?
    public var edited: Bool?
    public var editedby moderator: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var active: Bool?
    public var shadowban: Bool?
    public var likecount: Int64?
    public var replycount: Int64?
    public var reactions: [ChatEventReaction]
    public var moderation: String?
    public var reports: [ChatEventReport]
}

```

## 14.30 React to an Event

```

func reactToEvent(_ request: ChatRequest.ReactToEvent, completionHandler: @escaping _
↳ Completion<Event>)

```

Adds or removes 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.

### Parameters

- userid: (required) user id specific to app
- roomid: (required) Room Id, in which you want to react
- eventid: (required) message id, that you want to report.
- reacted: (required) true/false
- reaction: (required) e.g. like

**Warning** This method requires authentication.

**Request Model:** ChatRequest.ReactToEvent

```
public class ReactToEvent {  
    public var roomid: String?  
    public var eventid: String?  
    public var userid: String?  
    public var reaction: String?  
    public var reacted: String? = "false"  
}
```

**Response Model:** Event

```
open class Event: Codable, Equatable {  
    public var kind: String?  
    public var id: String?  
    public var roomid: String?  
    public var body: String?  
    public var originalbody: String?  
    public var added: Date?  
    public var modified: Date?  
    public var ts: Date?  
    public var eventtype: EventType?  
    public var userid: String?  
    public var user: User?  
    public var customtype: String?  
    public var customid: String?  
    public var custompayload: String?  
    public var customtags: [String]?  
    public var customfield1: String?  
    public var customfield2: String?  
    public var replyto: Event?  
    public var parentid: String?  
    public var edited: Bool?  
    public var editedbymoderator: Bool?  
    public var censored: Bool?  
    public var deleted: Bool?  
    public var active: Bool?  
    public var shadowban: Bool?  
    public var likecount: Int64?  
    public var replycount: Int64?  
    public var reactions: [ChatEventReaction]  
    public var moderation: String?  
    public var reports: [ChatEventReport]  
}
```

## 14.31 Report User in Room

```
func reportUserInRoom(_ request: ChatRequest.ReportUserInRoom, completionHandler: @escaping Completion<ChatRoom>)
```

Reports a user in the room

- This API enables users to report other users who exhibit abusive behaviors. It enables users to silence another user when a moderator is not present. If the user receives too many reports in a trailing 24 hour period, the user will become flagged at the room level.
- This API moderates users on the ROOM LEVEL. If there is an API method that enables reporting users at the global user level which impacts all rooms. This API impacts only the experience for the specified userid within the specified room.
- This API will return an error (see responses below) if user reporting is not enabled for your application in the application settings by setting User Reports limit to a value > 0.
- A user who is flagged will have the *shadowban* effect applied.

### Parameters

- roomid: (required) the id of the room in which you want to report the event
- userid: (required) the application specific user ID of the user reporting the first user
- reporttype: (required) [defaults="abuse"] Possible values: [.abuse, .spam]. SPAM is unsolicited commercial messages and abuse is hate speech or other unacceptable behavior.

### RESPONSE CODES

200 | OK : Sweet, sweet success.

404 | Not Found : The specified user or application could not be found.

412 | PreconditionFailed : The request was rejected because user reporting is not enabled for the application.

### Request Model: ChatRequest.ReportUserInRoom

```
public class ReportUserInRoom {
    public var roomid: String?
    public var userid: String?
    public var reporttype: ReportType? = .abuse
}
```

### Response Model: ChatRoom

```
public var kind: String?
public var id: String?
public var appid: String?
public var ownerid: String?
public var name: String?
public var description: String?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
```

(continues on next page)

(continued from previous page)

```

public var enableactions: Bool?
public var enableenterandexit: Bool?
public var open: Bool?
public var inroom: Int?
public var moderation: String?
public var maxreports: Int64?
public var enableprofanityfilter: Bool?
public var enableautoexpiresessions: Bool?
public var delaymessagesseconds: Int64?
public var added: Date?
public var whenmodified: Date?
public var bouncedusers: [String] = []
public var reportedusers: [ReportedUser] = []
}

```

## 14.32 Bounce User

```

func bounceUser(_ request: ChatRequest.BounceUser, completionHandler: @escaping _
↳ Completion<BounceUserRequest>)

```

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)

### Parameters

- userid: (required) user id specific to app
- bounce: (required) True if the user is being bounced from the room. False if user is debounced, allowing the user to reenter the room.
- roomid: (required) The ID of the chat room from which to bounce this user
- announcement: (optional) If provided, this announcement is displayed to the people who are in the room, as the body of a BOUNCE event.

### Request Model: ChatRequest.BounceUser

```

public class BounceUser {
    public var userid: String?
    public var bounce: Bool?
    public var roomid: String?
    public var announcement: String?
}

```

### Response Model: BounceUserResponse

```

public struct BounceUserResponse: Codable {
    public var kind: String?
    public var event: Event?
}

```

(continues on next page)



(continued from previous page)

```

    public var room: ChatRoom?
}

```

## 14.33 Shadowban User

```

func shadowbanUser(_ request: ChatRequest.ShadowbanUser, completionHandler: @escaping _
↳ Completion<ChatRoom>)

```

Shadow Ban User (In Room Only)

Will toggle the user's shadow banned flag.

There is a user level shadow ban (global) and local room level shadow ban.

A Shadow Banned user can send messages into a chat room, however those messages are flagged as shadow banned. This enables the application to show those messages only to the shadow banned user, so that that person may not know they were shadow banned. This method shadow bans the user on the global level (or you can use this method to lift the ban). You can optionally specify an expiration time. If the expiration time is specified, then each time the shadow banned user tries to send a message the API will check if the shadow ban has expired and will lift the ban.

### Parameters

- userid: (required) The applicaiton provided userid of the user to ban.
- applyeffect: (required) true or false. If true, user will be set to banned state. If false, will be set to non-banned state.
- expireseconds: (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

**Request Model: ChatRequest.ShadowbanUser**

```

public class ShadowbanUser {
    public var userid: String?
    public var roomid: String?
    public var applyeffect: Bool?
    public var expireseconds: Double?
}

```

**Response Model: ChatRoom**

```

public var kind: String?
    public var id: String?
    public var appid: String?
    public var ownerid: String?
    public var name: String?
    public var description: String?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?

```

(continues on next page)

(continued from previous page)

```

public var enableactions: Bool?
public var enableenterandexit: Bool?
public var open: Bool?
public var inroom: Int?
public var moderation: String?
public var maxreports: Int64?
public var enableprofanityfilter: Bool?
public var enableautoexpiresessions: Bool?
public var delaymessageseconds: Int64?
public var added: Date?
public var whenmodified: Date?
public var bouncedusers: [String] = []
public var reportedusers: [ReportedUser] = []
}

```

## 14.34 Mute User

```

func muteUser(_ request: ChatRequest.MuteUser, completionHandler: @escaping Completion
    <ChatRoom>)

```

Mute User (In Room Only)

Will toggle the user's shadow banned flag.

There is a user level shadow ban (global) and local room level shadow ban.

A Shadow Banned user can send messages into a chat room, however those messages are flagged as shadow banned. This enables the application to show those messages only to the shadow banned user, so that that person may not know they were shadow banned. This method shadow bans the user on the global level (or you can use this method to lift the ban). You can optionally specify an expiration time. If the expiration time is specified, then each time the shadow banned user tries to send a message the API will check if the shadow ban has expired and will lift the ban.

### Parameters

- userid: (required) The applicaiton provided userid of the user to ban.
- applyeffect: (required) true or false. If true, will have the mute affect applied. If false, mute will not be applied.
- expireseconds: (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

**Request Model:** ChatRequest.MuteUser

```

public class MuteUser {
    public var userid: String?
    public var roomid: String?
    public var applyeffect: Bool?
    public var expireseconds: Double?
}

```

**Response Model:** ChatRoom

```

public var kind: String?
    public var id: String?
    public var appid: String?
    public var ownerid: String?
    public var name: String?
    public var description: String?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var enableactions: Bool?
    public var enableenterandexit: Bool?
    public var open: Bool?
    public var inroom: Int?
    public var moderation: String?
    public var maxreports: Int64?
    public var enableprofanityfilter: Bool?
    public var enableautoexpiresessions: Bool?
    public var delaymessageseconds: Int64?
    public var added: Date?
    public var whenmodified: Date?
    public var bouncedusers: [String] = []
    public var reportedusers: [ReportedUser] = []
}

```

## 14.35 Search Event History

```

func searchEventHistory(_ request: ChatRequest.SearchEvent, completionHandler: @escaping _
↳ Completion<ListEventsResponse>)

```

Searches the message history applying the specified filters.

This returns displayable messages (for example speech, quote, threadedreply) that are in the active state (not flagged by moderator or logically deleted).

### Parameters

- fromuserid: (optional) Return only events from the specified user
- fromhandle: (optional) Return only events from a user with the specified handle. Exact match, case insensitive.
- roomid: (optional) Return only events in the specified room.
- body: (optional) Returns only messages which contain the specified body substring.
- limit: (optional) Default is 50, maximum is 200. Limits how many items are returned.
- cursor: (optional) Leave blank to start from the beginning of the result set; provide the value from the previous returned cursor to resume cursoring through the next page of results.
- direction: (optional) Defaults to Backward. Pass forward or backward. Backward is newest to oldest order, forward is oldest to newest order.
- types: (optional) Default = all. Use this to filter for specific event types.

- speech
- quote
- reply
- announcement
- custom
- reaction
- action
- enter
- exit
- ad
- roomopened
- roomclosed
- purge
- remove
- replace
- bounce

**Request Model: ChatRequest.SearchEvent**

```
public class SearchEvent {  
    public var fromuserid: String?  
    public var fromhandle: String?  
    public var roomid: String?  
    public var body: String?  
    public var limit: Int? = 50  
    public var cursor: String?  
    public var direction: Ordering?  
    public var types: [EventType]?  
}
```

**Response Model: Event**

```
public struct ListEventsResponse: Codable {  
    public var kind: String?  
    public var cursor: String?  
    public var more: Bool?  
    public var itemcount: Int64?  
    public var events: [Event]  
}
```

## 14.36 Update Chat Event

```
func updateChatEvent(_ request: ChatRequest.UpdateChatEvent, completionHandler: @escaping Completion<Event>)
```

Updates the contents of an existing chat event

This API may be used to update the body of an existing Chat Event. It is used to enable the user to edit the message after it is published. This may only be used with MESSAGE event types (speech, quote, reply). When the chat event is updated another event of type “replace” will be emitted with the updated event contents, and the original event will be replaced in future calls to List Event History, Join and List Previous Events. The event will also be flagged as edited by user.

### Parameters

- roomid: (required) The ID of the chat room conversation
- eventid: (required) The unique ID of the chat event to be edited. This must be a message type event (speech, quote or reply).
- userid: (required) The application specific user ID updating the chat event. This must be the owner of the comment or moderator / admin.
- body: (required) The new body contents of the event.
- customid: (optional) Optionally replace the customid.
- custompayload: (optional) Optionally replace the payload of the event.
- customfield1: (optional) Optionally replace the customfield1 value.
- customfield2: (optional) Optionally replace the customfield2 value.
- customtags: (optional) Optionally replace the custom tags.

### Request Model: ChatRequest.UpdateChatEvent

```
public class UpdateChatEvent {
    public var roomid: String?
    public var eventid: String?
    public var userid: String?
    public var body: String?
    public var customid: String?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?
    public var customtags: String?
}
```

### Response Model: Event

```
open class Event: Codable, Equatable {
    public var kind: String?
    public var id: String?
    public var roomid: String?
    public var body: String?
    public var originalbody: String?
    public var added: Date?
    public var modified: Date?
```

(continues on next page)

(continued from previous page)

```

public var ts: Date?
public var eventtype: EventType?
public var userid: String?
public var user: User?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var replyto: Event?
public var parentid: String?
public var edited: Bool?
public var editedbymoderator: Bool?
public var censored: Bool?
public var deleted: Bool?
public var active: Bool?
public var shadowban: Bool?
public var likecount: Int64?
public var replycount: Int64?
public var reactions: [ChatEventReaction]
public var moderation: String?
public var reports: [ChatEventReport]
}

```

## 14.37 Start Listening to Chat Updates

```

func startListeningToChatUpdates(config: ChatRequest.StartListeningToChatUpdates,
    ↪completionHandler: @escaping Completion<[Event]>)

```

Periodically calls func getUpdates(request:completionHandler:) to receive latest chat events.

### Parameters

- limit: (optional) Number of events to return. Default is 100, maximum is 500. Will use default if value set is below default value.
- eventSpacingMs: (optional) The frequency (in milliseconds) when events are dispatched from buffer. Will use default if value set is below default value.

### Request Model: ChatRequest.StartListeningToChatUpdates

```

public class StartListeningToChatUpdates {
    public var roomid: String    // REQUIRED
    public var limit: Int?
    public var eventSpacingMs: Int
}

```

### Response Model: Event

```

open class Event: Codable, Equatable {
    public var kind: String?
}

```

(continues on next page)

(continued from previous page)

```
public var id: String?
public var roomid: String?
public var body: String?
public var originalbody: String?
public var added: Date?
public var modified: Date?
public var ts: Date?
public var eventtype: EventType?
public var userid: String?
public var user: User?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var replyto: Event?
public var parentid: String?
public var edited: Bool?
public var editedbymoderator: Bool?
public var censored: Bool?
public var deleted: Bool?
public var active: Bool?
public var shadowban: Bool?
public var likecount: Int64?
public var replycount: Int64?
public var reactions: [ChatEventReaction]
public var moderation: String?
public var reports: [ChatEventReport]
}
```

## 14.38 Stop Listening to Chat Updates

```
func stopListeningToChatUpdates(_ roomid: String)
```

Cancels listening to Chat Updates from a specific ChatRoom

**Request Model:** None

**Response Model:** None

## 14.39 Approve Event

```
func approveEvent(_ request: ModerationRequest.ApproveEvent, completionHandler: @escaping Completion<Event>)
```

Approves a message in the moderation queue

If PRE-MODERATION is enabled for a room, then all messages go to the queue before they can appear in the event stream. For each incoming message, a webhook will be fired, if one is configured.

If the room is set to use POST-MODERATION, messages will only be sent to the moderation queue if they are reported.

**Warning** Requires Authentication

**Request Model:** ModerationRequest.ApproveEvent

```
public class ApproveEvent {
    public var roomid: String?
    public var eventid: String?
}
```

**Response Model:** Event

```
open class Event: Codable, Equatable {
    public var kind: String?
    public var id: String?
    public var roomid: String?
    public var body: String?
    public var originalbody: String?
    public var added: Date?
    public var modified: Date?
    public var ts: Date?
    public var eventtype: EventType?
    public var userid: String?
    public var user: User?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var replyto: Event?
    public var parentid: String?
    public var edited: Bool?
    public var editedbymoderator: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var active: Bool?
    public var shadowban: Bool?
    public var likecount: Int64?
    public var replycount: Int64?
    public var reactions: [ChatEventReaction]
    public var moderation: String?
    public var reports: [ChatEventReport]
}
```



## 14.40 Reject Event

```
func rejectEvent(_ request: ModerationRequest.RejectEvent, completionHandler: @escaping _
↳Completion<Event>)
```

Rejects a message in the moderation queue

If PRE-MODERATION is enabled for a room, then all messages go to the queue before they can appear in the event stream. For each incoming message, a webhook will be fired, if one is configured.

If the room is set to use POST-MODERATION, messages will only be sent to the moderation queue if they are reported.

**Warning** This method requires authentication

**Request Model:** ModerationRequest.RejectEvent

```
public class RejectEvent {
    public var roomid: String?
    public var eventid: String?
}
```

**Response Model:** Event

```
open class Event: Codable, Equatable {
    public var kind: String?
    public var id: String?
    public var roomid: String?
    public var body: String?
    public var originalbody: String?
    public var added: Date?
    public var modified: Date?
    public var ts: Date?
    public var eventtype: EventType?
    public var userid: String?
    public var user: User?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var replyto: Event?
    public var parentid: String?
    public var edited: Bool?
    public var editedbymoderator: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var active: Bool?
    public var shadowban: Bool?
    public var likecount: Int64?
    public var replycount: Int64?
    public var reactions: [ChatEventReaction]
    public var moderation: String?
    public var reports: [ChatEventReport]
}
```

## 14.41 List All Messages In Moderation Queue

```
func listMessagesInModerationQueue(_ request: ModerationRequest.  
↳listMessagesInModerationQueue, completionHandler: @escaping Completion  
↳<ListMessagesNeedingModerationResponse>)
```

List all the messages in the moderation queue

### Parameters

- limit: (optional) Defaults to 200. This limits how many messages to return from the queue
- roomId: (optional) Provide the ID for a room to filter for only the queued events for a specific room
- cursor: (optional) Provide cursor value to get the next page of results.

**Warning** This method requires authentication

**Request Model:** ModerationRequest.listMessagesInModerationQueue

```
public class listMessagesInModerationQueue {  
    public var limit: Int? = 200  
    public var roomId: String?  
    public var cursor: String?  
}
```

**Response Model:** ListMessagesNeedingModerationResponse

```
public struct ListMessagesNeedingModerationResponse: Codable {  
    public var kind: String?  
    public var events: [Event]  
}
```

## COMMENT CLIENT

### 15.1 Create or Update Conversation

```
func createOrUpdateConversation(_ request: CommentRequest.CreateUpdateConversation, ↵  
    ↵completionHandler: @escaping Completion<Conversation>)
```

Creates a conversation (a context for comments)

#### Parameters

- **conversationid** : (optional) The conversation ID. This must be a URL friendly string (cannot contain / ? or other URL delimiters). Maximum length is 250 characters.
- **property** : (required) The property this conversation is associated with. It is any string value you want. Typically this is the domain of your website for which you want to use commenting, if you have more than one. Examples: (“dev”, “uat”, “stage”, “prod”, “site1.com”, “site2.com”)
- **moderation**  
[(required) Specify if pre or post moderation is to be used]
  - *pre* - marks the room as Premoderated
  - *post* - marks the room as Postmoderated
- **maxreports** : (optional, default = 3) If this number of users flags a content item in this conversation, the item is disabled and sent to moderator queue for review
- **enableprofanityfilter**: (optional) [default=true / false] Enables profanity filtering.
- **title** : (optional) The title of the conversation
- **maxcommentlen**: (optional) The maximum allowed length of a comment. Default is 256 characters. Maximum value is 10485760 (10 MB)
- **open**: (optional, defaults to true) If the conversation is open people can add comments.
- **added**: (optional) If this timestamp is provided then the whenadded field will be overridden. You should only use this when migrating data; data is timestamped automatically. Example value: “2020-05-02T08:51:53.8140055Z”
- **whenmodified**: (optional)
- **customtype** : (optional) Custom type string.
- **customid**: (optional) 250 characters for a custom ID for your app. This field is indexed for high performance object retrieval.
- **customtags** : (optional) A comma delimited list of tags

- **custompayload** : (optional) Custom payload string.
- **customfield1** : (optional) User custom field 1. Store any string value you want here, limit 1024 bytes.
- **customfield2** : (optional) User custom field 2. Store any string value you want here, limit 1024 bytes.

**Warning** This method requires authentication

**Request Model: CommentRequest.CreateUpdateConversation**

```
public class CreateUpdateConversation: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String?
    public let property: String           // REQUIRED
    public let moderation: String         // REQUIRED
    public var maxreports: Int?
    public var enableprofanityfilter: Bool?
    public var title: String?
    public var maxcommentlen: Int64?
    public var open: Bool?
    public var added: String? // OPTIONAL, Example value: "2020-05-02T08:51:53.
    ↪8140055Z"
    public var whenmodified: String? // OPTIONAL, Example value: "2020-05-
    ↪02T08:51:53.8140055Z"
    public var customtype: String?
    public var customid: String?
    public var customtags: [String]?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?
    /// ...
}
```

**Response Model: Conversation**

```
open class Conversation: Codable {
    public var kind: String? // "comment.conversation"
    public var id: String?
    public var appid: String?
    public var owneruserid: String?
    public var conversationid: String?
    public var property: String? // "sportstalk247.com/apidemo"
    public var moderation: String? /* "pre"/"post"/"na" */
    public var maxreports: Int64? // OPTIONAL, defaults to 3
    public var enableprofanityfilter: Bool? // OPTIONAL, defaults to true
    public var title: String?
    public var maxcommentlen: Int64?
    public var commentcount: Int64?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var open: Bool? // OPTIONAL, defaults to true
    public var added: Date? // OPTIONAL, Example value: "2020-05-02T08:51:53.
    ↪8140055Z"
    public var whenmodified: Date? // OPTIONAL, Example value: "2020-05-
    ↪02T08:51:53.8140055Z"
    public var customtype: String?
```

(continues on next page)

(continued from previous page)

```

public var customid: String?
public var customtags: [String]?
public var custompayload: String?
public var customfield1: String?
public var customfield2: String?

    /// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.CreateUpdateConversation(
    conversationid: "demo-test-conversation-1",
    property: "sportstalk247.com/apidemo1",
    moderation: "post"
)
request.enableprofanityfilter = false
request.title = "Sample Conversation 1"
request.open = true
request.customid = "test-custom-convo-id1"
// Other request optional fields

// Perform operation
commentClient.createOrUpdateConversation(request) { (code: Int?, message: String?, kind:
↳String?, response: Conversation?) in
    // ... Resolve `response` from here
}

```

## 15.2 Get Conversation by ID

```

func getConversation(_ request: CommentRequest.GetConversationById, completionHandler:
↳@escaping Completion<Conversation>)

```

Retrieves metadata about a conversation.

**Parameters**

- **conversationid** : (required) The ID of the conversation which is a context for comments. The ID must be URL ENCODED.

**Warning** This method requires authentication

**Request Model:** CommentRequest.GetConversationById

```

public class GetConversationById: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String    // REQUIRED
    /// ...
}

```

**Response Model:** Conversation

```

open class Conversation: Codable {
    public var kind: String? // "comment.conversation"
    public var id: String?
    public var appid: String?
    public var owneruserid: String?
    public var conversationid: String?
    public var property: String? // "sportstalk247.com/apidemo"
    public var moderation: String? /* "pre"/"post"/"na" */
    public var maxreports: Int64? // OPTIONAL, defaults to 3
    public var enableprofanityfilter: Bool? // OPTIONAL, defaults to true
    public var title: String?
    public var maxcommentlen: Int64?
    public var commentcount: Int64?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var open: Bool? // OPTIONAL, defaults to true
    public var added: Date? // OPTIONAL, Example value: "2020-05-02T08:51:53.
↳8140055Z"
    public var whenmodified: Date? // OPTIONAL, Example value: "2020-05-
↳02T08:51:53.8140055Z"
    public var customtype: String?
    public var customid: String?
    public var customtags: [String]?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?

    /// ...
}

```

### Example

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.GetConversationById(conversationid: "conversation-id-1")

// Perform operation
commentClient.getConversation(request) { (code: Int?, message: String?, kind: String?,
↳response: Conversation?) in
    // ... Resolve `response` from here
}

```

## 15.3 Find Conversation by CustomID

```

func getConversationByCustomId(_ request: CommentRequest.FindConversationByIdCustomId,
↳completionHandler: @escaping Completion<Conversation>)

```

Uses the CustomID for the conversation supplied by the app to retrieve the conversation object. It returns exactly one object or 404 if not found. This query is covered by an index and is performant.

### Parameters

- **customid** : (Required) Locates a conversation using the custom ID.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.FindConversationByIdCustomId`

```
public class FindConversationByIdCustomId: ParametersBase</*...*/> {
    /// ...
    public let customid: String    // REQUIRED
    /// ...
}
```

**Response Model:** `Conversation`

```
open class Conversation: Codable {
    public var kind: String?    // "comment.conversation"
    public var id: String?
    public var appid: String?
    public var owneruserid: String?
    public var conversationid: String?
    public var property: String?    // "sportstalk247.com/apidemo"
    public var moderation: String? /* "pre"/"post"/"na" */
    public var maxreports: Int64?    // OPTIONAL, defaults to 3
    public var enableprofanityfilter: Bool? // OPTIONAL, defaults to true
    public var title: String?
    public var maxcommentlen: Int64?
    public var commentcount: Int64?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var open: Bool?    // OPTIONAL, defaults to true
    public var added: Date?    // OPTIONAL, Example value: "2020-05-02T08:51:53.
    ↪8140055Z"
    public var whenmodified: Date?    // OPTIONAL, Example value: "2020-05-
    ↪02T08:51:53.8140055Z"
    public var customtype: String?
    public var customid: String?
    public var customtags: [String]?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?

    /// ...
}
```

**Example**

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.FindConversationByIdCustomId(customid: "custom-conversation-
    ↪idl")

// Perform operation
commentClient.getConversationByCustomId(request) { (code: Int?, message: String?, kind:
    ↪String?, response: Conversation?) in
    // ... Resolve `response` from here
}
```

## 15.4 List Conversations

```
func listConversations(_ request: CommentRequest.ListConversations, completionHandler: @escaping Completion<ListConversationsResponse>)
```

Retrieves metadata about all conversations for a property. Whenever you create a conversation, you provide a property to associate it with. This returns the metadata for all conversations associated with a property.

### ABOUT CURSORING:

- API Method returns a cursor
- Cursor includes a “more” field indicating if there are more results that can be read at the time this call is made
- Cursor includes “cursor” field, which can be passed into subsequent calls to this method to get additional results
- Cursor includes “itemcount” field, which is the number of items returned by the cursor not the total number of items in the database
- All LIST methods in the API return cursors and they all work the same way

### Parameters

- **propertyid** : Filters list of conversations by property. Exact match only, case sensitive.
- **cursor** : (Optional, default = “”). For cursoring, pass in cursor output from previous call to continue where you left off.
- **limit** : (Optional, default = 200). For cursoring, limit the number of responses for this request.
- **sort**  
 [(Optional, default = “oldest”).]
  - *newest* : Default. Sorts from newest created conversation to the oldest.
  - *oldest* : Starts from oldest conversation and cursors towards the newest.

**Warning** This method requires authentication

### Request Model: CommentRequest.ListConversations

```
public class ListConversations: ParametersBase</*...*/> {
    /// ...
    public var propertyid: String?
    public var cursor: String?
    public var limit: Int?
    public var sort: SortType?
    /// ...
}
```

### Response Model: ListConversationsResponse

```
open class ListConversationsResponse: Codable {
    public var kind: String? /* "list.commentconversations" */
    public var cursor: String?
    public var more: Bool?
    public var conversations: [Conversation] = []

    /// ...
}
```



### Example

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.ListConversations()
// You may provide optional parameters as shown below:
// request.propertyid = "sportstalk247.com/apidemo"
// request.cursor = "63bd442ccfce070c7825639a"
// request.limit = 10
// request.sort = SortType.newest

// Perform operation
commentClient.listConversations(request) { (code: Int?, message: String?, kind: String?,
↳response: ListConversationsResponse?) in
    // ... Resolve `response` from here
}

```

## 15.5 Batch Get Conversation Details

```

func batchGetConversationDetails(_ request: CommentRequest.GetBatchConversationDetails,
↳completionHandler: @escaping Completion<BatchGetConversationDetailsResponse>)

```

The purpose of this method is to support a use case where you start with a list of conversations and you want metadata about only those conversations so you can display things like like count or comment count making minimal requests. You can choose to either retrieve articles using the sportstalk ID or by using your custom IDs you associated with the conversation using our create/update conversation API.

### Parameters

- **ids** : (optional): Include one or more comma delimited Sportstalk conversation IDs.
- **cid** : (optional): Include one or more cid arguments. Each is a URL ENCODED string containing the customid. You can specify up to 200 at a time.
- **entities (optional):** By default only the conversation object data is returned. For more data (and deeper queries) provide any of these entities:
  - *reactions*: Includes user reactions and microprofiles in the response
  - *likecount*: Includes number of likes on the conversation in the response, otherwise returns -1 for like count.
  - *commentcount*: Includes the number of comments in the response, otherwise returns -1 for comment count.

**Warning** This method requires authentication

**Request Model:** CommentRequest.GetBatchConversationDetails

```

public class GetBatchConversationDetails: ParametersBase</*...*/> {
    /// ...
    public var ids: [String]?
    public var cid: [String]?
    public var entities: [BatchGetConversationEntity]?
    /// ...
}

```

**Response Model:** BatchGetConversationDetailsResponse

```
open class BatchGetConversationDetailsResponse: Codable {
    public var kind: String? /* "list.comment.conversation.details" */
    public var itemcount: Int64?
    public var conversations: [Conversation] = []

    /// ...
}
```

### Example

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.GetBatchConversationDetails(
    ids: [createdConversation1.conversationid!, createdConversation2.conversationid!]
)
//
// Optionally, if NOT `ids`, you may either provided customids under `cid` parameter
// let request = CommentRequest.GetBatchConversationDetails(
//     cid: [createdConversation1.conversationid!, createdConversation2.
// ↪ conversationid!]
// )
//
// Lastly, you may also provide `entities` parameter to include more data

// Perform operation
commentClient.batchGetConversationDetails(request) { (code: Int?, message: String?, ↪
// ↪ kind: String?, response: BatchGetConversationDetailsResponse?) in
//     // ... Resolve `response` from here
}
```

## 15.6 React to Conversation Topic

```
func reactToConversationTopic(_ request: CommentRequest.ReactToConversationTopic, ↪
// ↪ completionHandler: @escaping Completion<Conversation>)
```

Adds or removes a reaction to a topic A conversation context is mapped to your topic by using either the conversationid or the customid. You can either react to the content itself (for example to LIKE an article/video/poll) or you can use the comment react api to react to an individual comment. This method is for commenting on the conversation topic level.

### Parameters

- **userid** : (required) The ID of the user reacting to the comment. Anonymous reactions are not supported.
- **reaction** : (required) A string indicating the reaction you wish to capture, for example “like”, or “emoji:{id}” where you can use the standard character code for your emoji.
- **reacted** : (required) true or false, to toggle the reaction on or off for this user.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.ReactToConversationTopic`

```
public class ReactToConversationTopic: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
```

(continues on next page)

(continued from previous page)

```

public let userid: String    // REQUIRED
public let reaction: String // REQUIRED
public let reacted: Bool     // REQUIRED
    /// ...
}

```

**Response Model: Conversation**

```

open class Conversation: Codable {
    public var kind: String?    // "comment.conversation"
    public var id: String?
    public var appid: String?
    public var owneruserid: String?
    public var conversationid: String?
    public var property: String? // "sportstalk247.com/apidemo"
    public var moderation: String? /* "pre"/"post"/"na" */
    public var maxreports: Int64? // OPTIONAL, defaults to 3
    public var enableprofanityfilter: Bool? // OPTIONAL, defaults to true
    public var title: String?
    public var maxcommentlen: Int64?
    public var commentcount: Int64?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var open: Bool? // OPTIONAL, defaults to true
    public var added: Date? // OPTIONAL, Example value: "2020-05-02T08:51:53.
    ↪8140055Z"
    public var whenmodified: Date? // OPTIONAL, Example value: "2020-05-
    ↪02T08:51:53.8140055Z"
    public var customtype: String?
    public var customid: String?
    public var customtags: [String]?
    public var custompayload: String?
    public var customfield1: String?
    public var customfield2: String?

    /// ...
}

```

**Example**


```

let commentClient = CommentClient(config: config)
let request = CommentRequest.ReactToConversationTopic(
    conversationid: createdConversation.conversationid!,
    userid: createdUser.userid!,
    reaction: "like",
    reacted: true
)

// Perform operation
commentClient.reactToConversationTopic(request) { (code: Int?, message: String?, kind:
    ↪String?, response: Conversation?) in
    // ... Resolve `response` from here
}

```

## 15.7 Create and Publish Comment

```
func createComment(_ request: CommentRequest.CreateComment, completionHandler: @escaping  Completion<Comment>)
```

Creates a comment and publishes it. You can optionally make this comment into a reply by passing in the optional `replyto` field. Custom fields can be set, and can be overwritten. However, once a custom field is used it can not be set to no value (empty string).

### Parameters

- **conversationid** : (required) The ID of the comment stream to publish the comment to. See the Create / Update Conversation method for rules around `conversationid`.
- **userid** : (required) The application's `userid` representing the user who submitted the comment
- **displayname** : (optional). This is the desired name to display, typically the real name of the person.
- **body** : (required) The body of the comment (the message). Supports unicode characters including EMOJIs and international characters.
- **customtype** : (optional) Custom type string.
- **customfield1** : (optional) User custom field 1. Store any string value you want here, limit 1024 bytes.
- **customfield2** : (optional) User custom field 2. Store any string value you want here, limit 1024 bytes.
- **customtags** : (optional) A comma delimited list of tags
- **custompayload** : (optional) Custom payload string.

**Warning** This method requires authentication

### Request Model: CommentRequest.CreateComment

```
public class CreateComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let userid: String // REQUIRED
    public var displayname: String?
    public let body: String // REQUIRED
    public var customtype: String?
    public var customfield1: String?
    public var customfield2: String?
    public var customtags: [String]?
    public var custompayload: String?
    /// ...
}
```

### Response Model: Comment

```
open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
    public var commenttype: String? // "comment"
    public var added: Date?
    public var modified: Date?
```

(continues on next page)

(continued from previous page)

```

public var tsunix: Int64?
public var userid: String?
public var user: User?
public var body: String?
public var originalbody: String?
public var hashtags: [String]?
public var shadowban: Bool?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var edited: Bool?
public var censored: Bool?
public var deleted: Bool?
public var parentid: String?
public var hierarchy: [String]?
public var reactions: [Reaction]?
public var likecount: Int64?
public var replycount: Int64?
public var votecount: Int64?
public var votescore: Int64?
public var votes: [Vote]?
public var moderation: String? // "approved", "pending", "rejected"
public var active: Bool?
public var reports: [Report]?

/// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.CreateComment(
    conversationid: "demo-conversation-id",
    userid: "yBommvwYYNBrxPwY",
    body: "Sample Comment 1"
)
// Optionally, you may provide more parameters
// request.displayname = "MyAlterEgo1"
// request.customtype = "type1"
// request.customfield1 = "/sample/userdefined1/emojis/"
// request.customfield2 = "/sample/userdefined2/intl/characters/"
// request.customtags = ["taga", "tagb"]
// request.custompayload = "{ num: 0 }"

// Perform operation
commentClient.createComment(request) { (code: Int?, message: String?, kind: String?,
↳ response: Comment?) in
    // ... Resolve `response` from here
}

```

## 15.8 Reply to Comment

```
func replyToComment(_ request: CommentRequest.ReplyToComment, completionHandler: @escaping Completion<Comment>)
```

Creates a reply to a comment and publishes it

The reply to comment method is the same as the create comment method, except you pass in the ID of the parent comment using the replyto field. See WEBHOOKS SERVICE API for information on receiving a notification when someone replies to a comment. See documentation on Create and Publish Comment

### Parameters

- **conversationid** : (required) The ID of the comment conversation.
- **replytocommentid** : (required) The unique ID of the comment we will reply to.
- **userid** : (required) The application's userid representing the user who submitted the comment
- **displayname** : (optional). This is the desired name to display, typically the real name of the person.
- **body** : (required) The body of the reply (what the user is saying). Supports unicode characters including EMOJIs and international characters.
- **customtype** : (optional) Custom type string.
- **customfield1** : (optional) User custom field 1. Store any string value you want here, limit 1024 bytes.
- **customfield2** : (optional) User custom field 2. Store any string value you want here, limit 1024 bytes.
- **customtags** : (optional) A comma delimited list of tags
- **custompayload** : (optional) Custom payload string.

**Warning** This method requires authentication

### Request Model: CommentRequest.ReplyToComment

```
public class ReplyToComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let replytocommentid: String // REQUIRED
    public let userid: String // REQUIRED
    public var displayname: String?
    public let body: String // REQUIRED
    public var customtype: String?
    public var customfield1: String?
    public var customfield2: String?
    public var customtags: [String]?
    public var custompayload: String?
    /// ...
}
```

### Response Model: Comment

```
open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
```

(continues on next page)

(continued from previous page)

```

public var commenttype: String? // "comment"
public var added: Date?
public var modified: Date?
public var tsunix: Int64?
public var userid: String?
public var user: User?
public var body: String?
public var originalbody: String?
public var hashtags: [String]?
public var shadowban: Bool?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var edited: Bool?
public var censored: Bool?
public var deleted: Bool?
public var parentid: String?
public var hierarchy: [String]?
public var reactions: [Reaction]?
public var likecount: Int64?
public var replycount: Int64?
public var votecount: Int64?
public var votescore: Int64?
public var votes: [Vote]?
public var moderation: String? // "approved", "pending", "rejected"
public var active: Bool?
public var reports: [Report]?

/// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.ReplyToComment(
    conversationid: "demo-conversation-id1",
    replytocommentid: "root-comment-id1",
    userid: "yBommvwYYNBrxPwY",
    body: "Reply Comment"
)
// Optionally, you may provide more parameters
// request.displayname = "MyAlterEgo1"
// request.customtype = "type1"
// request.customfield1 = "/sample/userdefined1/emojis/"
// request.customfield2 = "/sample/userdefined2/intl/characters/"
// request.customtags = ["taga", "tagb"]
// request.custompayload = "{ num: 0 }"

// Perform operation

```

(continues on next page)

(continued from previous page)

```
commentClient.replyToComment(request) { (code: Int?, message: String?, kind: String?,  
↳response: Comment?) in  
    // ... Resolve `response` from here  
}
```

## 15.9 List Replies

```
func listCommentReplies(_ request: CommentRequest.ListCommentReplies, completionHandler: (↳  
↳@escaping Completion<ListCommentsResponse>)
```

Get a list of replies to a comment

This method works the same way as the List Comments method, so view the documentation on that method. The difference is that this method will filter to only include comments that have a parent.

### ABOUT CURSORING:

- API Method returns a cursor
- Cursor includes a “more” field indicating if there are more results that can be read at the time this call is made
- Cursor includes “cursor” field, which can be passed into subsequent calls to this method to get additional results
- Cursor includes “itemcount” field, which is the number of items returned by the cursor not the total number of items in the database
- All LIST methods in the API return cursors and they all work the same way

### Parameters

- **conversationid** : (required) The ID of the comment conversation.
- **cursor** : (optional) If provided, will get the next bundle of comments in the conversation resuming from where the cursor left off.
- **limit** : (Optional, default = 200). For cursoring, limit the number of responses for this request.
- **direction**: (optional) Default is forward. Must be forward or backward
- **sort**  
 [(optional, defaults to “oldest”) Specifies that sort should be done by...]
  - *oldest* : Sort by when added ascending (oldest on top)
  - *newest* : Sort by when added ascending (newest on top)
  - *likes* : Sort by number of likes, descending (most liked on top)
  - *votescore* : Sort by net of adding upvotes and subtracting downvotes, descending
  - *mostreplies* : Sort by number of replies, descending
- **includechildren** : (optional, default is false) If false, this returns all reply nodes that are immediate children of the provided parent id. If true, it includes all replies under the parent id and all the children of those replies and so on.
- **includeinactive** : (optional, default is false) If true, return comments that are inactive (for example, disabled by moderation)



**Warning** This method requires authentication

**Request Model:** `CommentRequest.ListCommentReplies`

```
public class ListCommentReplies: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    public var cursor: String?
    public var limit: Int?
    public var direction: Ordering?
    public var sort: SortType?
    public var includechildren: Bool?
    public var includeinactive: Bool?
    /// ...
}
```

**Response Model:** `ListCommentsResponse`

```
open class ListCommentsResponse: Codable {
    public var kind: String? /* "list.comments" */
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var conversation: Conversation?
    public var comments: [Comment] = []

    /// ...
}
```

**Example**

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.ListCommentReplies(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1"
)
// You may provide optional parameters as shown below:
// request.propertyid = "sportstalk247.com/apidemo"
// request.cursor = "63bd442ccfce070c7825639a"
// request.limit = 10
// request.sort = SortType.newest

// Perform operation
commentClient.listCommentReplies(request) { (code: Int?, message: String?, kind: String?,
↪ response: ListCommentsResponse?) in
    // ... Resolve `response` from here
}
```

## 15.10 Get Comment by ID

```
func getComment(_ request: CommentRequest.GetCommentDetails, completionHandler: @escaping Completion<Comment>)
```

The comment time stamp is stored in UTC time.

### Parameters

- **conversationid** : (required) The ID of the comment conversation, URL ENCODED.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.\*

**Warning** This method requires authentication

### Request Model: CommentRequest.GetCommentDetails

```
public class GetCommentDetails: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    /// ...
}
```

### Response Model: Comment

```
open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
    public var commenttype: String? // "comment"
    public var added: Date?
    public var modified: Date?
    public var tsunix: Int64?
    public var userid: String?
    public var user: User?
    public var body: String?
    public var originalbody: String?
    public var hashtags: [String]?
    public var shadowban: Bool?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var edited: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var parentid: String?
    public var hierarchy: [String]?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var replycount: Int64?
```

(continues on next page)

(continued from previous page)

```

public var voteCount: Int64?
public var voteScore: Int64?
public var votes: [Vote]?
public var moderation: String? // "approved", "pending", "rejected"
public var active: Bool?
public var reports: [Report]?

/// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.GetCommentDetails(
    conversationid: "demo-conversation-id1",
    commentid: "get-comment-id1"
)

// Perform operation
commentClient.getComment(request) { (code: Int?, message: String?, kind: String?,
↳ response: Comment?) in
    // ... Resolve `response` from here
}

```

## 15.11 List Comments

```

func listComments(_ request: CommentRequest.ListComments, completionHandler: @escaping
↳ Completion<ListCommentsResponse>)

```

Get a list of comments within a conversation

**ABOUT CURSORING:**

- API Method returns a cursor
- Cursor includes a “more” field indicating if there are more results that can be read at the time this call is made
- Cursor includes “cursor” field, which can be passed into subsequent calls to this method to get additional results
- Cursor includes “itemcount” field, which is the number of items returned by the cursor not the total number of items in the database
- All LIST methods in the API return cursors and they all work the same way

**Parameters**

- **conversationid** : (required) The ID of the comment conversation.
- **cursor** : (optional) If provided, will get the next bundle of comments in the conversation resuming from where the cursor left off.
- **limit** : (Optional, default = 200). For cursoring, limit the number of responses for this request.
- **direction**: (optional) Default is forward. Must be forward or backward
- **sort**  
[(optional, defaults to “oldest”) Specifies that sort should be done by...]

- *oldest* : Sort by when added ascending (oldest on top)
- *newest* : Sort by when added ascending (newest on top)
- *likes* : Sort by number of likes, descending (most liked on top)
- *votescore* : Sort by net of adding upvotes and subtracting downvotes, descending
- *mostreplies* : Sort by number of replies, descending
- **includechildren** : (optional, default is false) If false, this returns all reply nodes that are immediate children of the provided parent id. If true, it includes all replies under the parent id and all the children of those replies and so on.
- **includeinactive** : (optional, default is false) If true, return comments that are inactive (for example, disabled by moderation)

**Warning** This method requires authentication

#### Request Model: `CommentRequest.ListComments`

```
public class ListComments: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String    // REQUIRED
    public var cursor: String?
    public var limit: Int?
    public var direction: Ordering?
    public var sort: SortType?
    public var includechildren: Bool?
    public var includeinactive: Bool?
    /// ...
}
```

#### Response Model: `ListCommentsResponse`

```
open class ListCommentsResponse: Codable {
    public var kind: String?    /* "list.comments" */
    public var cursor: String?
    public var more: Bool?
    public var itemcount: Int64?
    public var conversation: Conversation?
    public var comments: [Comment] = []

    /// ...
}
```

#### Example

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.ListComments(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1"
)
// You may provide optional parameters as shown below:
// request.propertyid = "sportstalk247.com/apidemo"
// request.cursor = "63bd442ccfce070c7825639a"
// request.limit = 10
// request.sort = SortType.newest
```

(continues on next page)

(continued from previous page)

```
// Perform operation
commentClient.listComments(request) { (code: Int?, message: String?, kind: String?,
↳response: ListCommentsResponse?) in
    // ... Resolve `response` from here
}
```

## 15.12 List Replies Batch

```
func listCommentRepliesBatch(_ request: CommentRequest.GetBatchCommentReplies,
↳completionHandler: @escaping Completion<ListCommentRepliesBatchResponse>)
```

Get a list of replies to multiple parent Comments

The purpose of this method is to support a use case where you open an app or website widget and you have just displayed up to N top level comments and you want to retrieve the replies to those comments quickly, in 1 request. You could call GetReplies for each top level parent, but if you want to get them in just one request use this method, which has more speed but some limitations:

- This method does not support cursoring.
- This method allows you to specify the maximum number of children to return per top level parent, but it does not apply a limit across the total number of replies across all of the top level comments.
- This method will always return replies sorted by when originally published timestamp ascending (oldest to newest), with replies grouped by each parent comment in the result set.
- This method will return the children that are direct immediate child replies to the parent only, not an entire tree under a parent.
- If the parentid list contains a parentid that does not exist or has no child replies it will be skipped, you will not receive 404 unless none of the parentids were found.

### Parameters

- **conversationid** : (required) The ID of the comment conversation.
- **childlimit** : (Optional, default = 50).
- **parentids** : (Required). A list of parent comment ID(s), up to 30.
- **includeinactive** : (optional, default is false) If true, return comments that are inactive (for example, disabled by moderation)

**Warning** This method requires authentication

**Request Model:** `CommentRequest.GetBatchCommentReplies`

```
public class GetBatchCommentReplies: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String    // REQUIRED
    public var childlimit: Int?
    public let parentids: [String]    // REQUIRED
    public var includeinactive: Bool?
    /// ...
}
```

**Response Model: ListCommentRepliesBatchResponse**

```

open class ListCommentRepliesBatchResponse: Codable {
    public var kind: String?
    public var repliesgroupedbyparentid: [CommentReplyGroup]

    /// ...

    public struct CommentReplyGroup: Codable {
        public var kind: String?
        public var parentid: String?
        public var comments: [Comment] = []

        // ...
    }
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.GetBatchCommentReplies(
    conversationid: "demo-conversation-id1",
    parentids: ["root-comment-id1", "root-comment-id2", "root-comment-id3"]
)
// You may provide optional parameters as shown below:
// request.childlimit = 10
// request.includeinactive = true

// Perform operation
commentClient.listCommentRepliesBatch(request) { (code: Int?, message: String?, kind:
↳String?, response: ListCommentRepliesBatchResponse?) in
    // ... Resolve `response` from here
}

```

## 15.13 React to Comment(“Like”)

```

func reactToComment(_ request: CommentRequest.ReactToComment, completionHandler:
↳@escaping Completion<Comment>)

```

Adds or removes a reaction to a comment

A reaction can be added using any reaction string that you wish.

**Parameters**

- **conversationid** : (required) The ID of the comment conversation.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.\*
- **\*\*userid** : (required) The ID of the user reacting to the comment. Anonymous reactions are not supported.
- **reaction** : (required) A string indicating the reaction you wish to capture, for example “like”, or “emoji:{id}” where you can use the standard character code for your emoji.
- **reacted** : (required) true or false, to toggle the reaction on or off for this user.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.ReactToComment`

```
public class ReactToComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    public let userid: String // REQUIRED
    public let reaction: String // REQUIRED
    public let reacted: Bool // REQUIRED
    /// ...
}
```

**Response Model:** `Comment`

```
open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
    public var commenttype: String? // "comment"
    public var added: Date?
    public var modified: Date?
    public var tsunix: Int64?
    public var userid: String?
    public var user: User?
    public var body: String?
    public var originalbody: String?
    public var hashtags: [String]?
    public var shadowban: Bool?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var edited: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var parentid: String?
    public var hierarchy: [String]?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var replycount: Int64?
    public var votecount: Int64?
    public var votescore: Int64?
    public var votes: [Vote]?
    public var moderation: String? // "approved", "pending", "rejected"
    public var active: Bool?
    public var reports: [Report]?

    /// ...
}
```

## Example

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.ReactToComment(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1",
    userid: "yBommvwYYNBrxPwY",
    reaction: "like",
    reacted: true
)

// Perform operation
commentClient.reactToComment(request) { (code: Int?, message: String?, kind: String?,
↳ response: Comment?) in
    // ... Resolve `response` from here
}
```

## 15.14 Vote on Comment

```
func voteOnComment(_ request: CommentRequest.VoteOnComment, completionHandler: @escaping _
↳ Completion<Comment>)
```

UPVOTE, DOWNVOTE, or REMOVE VOTE

## Parameters

- **conversationid** : (required) The ID of the comment conversation.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.\*
- **vote** : (required) Must be one of “up”, “down”, or “none” (empty value). If up, the comment receives an upvote. If down, the comment receives a down vote. If empty, the vote is removed.
- **userid** : (required) The application specific user id performing the action.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.VoteOnComment`

```
public class VoteOnComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    public let vote: VoteType // REQUIRED
    public let userid: String // REQUIRED
    /// ...
}
```

**Response Model:** `Comment`

```
open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
```

(continues on next page)



(continued from previous page)

```

public var commenttype: String? // "comment"
public var added: Date?
public var modified: Date?
public var tsunix: Int64?
public var userid: String?
public var user: User?
public var body: String?
public var originalbody: String?
public var hashtags: [String]?
public var shadowban: Bool?
public var customtype: String?
public var customid: String?
public var custompayload: String?
public var customtags: [String]?
public var customfield1: String?
public var customfield2: String?
public var edited: Bool?
public var censored: Bool?
public var deleted: Bool?
public var parentid: String?
public var hierarchy: [String]?
public var reactions: [Reaction]?
public var likecount: Int64?
public var replycount: Int64?
public var votecount: Int64?
public var votescore: Int64?
public var votes: [Vote]?
public var moderation: String? // "approved", "pending", "rejected"
public var active: Bool?
public var reports: [Report]?

/// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.VoteOnComment(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1",
    vote: VoteType.up,
    userid: "yBommvwYYNBrxPwY"
)

// Perform operation
commentClient.voteOnComment(request) { (code: Int?, message: String?, kind: String?,
↳ response: Comment?) in
    // ... Resolve `response` from here
}

```

## 15.15 Report Comment

```
func reportComment(_ request: CommentRequest.ReportComment, completionHandler: @escaping _
↳ Completion<Comment>)
```

REPORTS a comment to the moderation team.

### Parameters

- **conversationid** : (required) The ID of the comment conversation.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.
- **userid** : (required) This is the application specific user ID of the user reporting the comment.
- **reporttype** : (required) A string indicating the reason you wish to report(i.e. “abuse”, “spam”).

**Warning** This method requires authentication

### Request Model: CommentRequest.ReportComment

```
public class ReportComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    public let userid: String // REQUIRED
    public let reporttype: ReportType // REQUIRED
    /// ...
}
```

### Response Model: Comment

```
open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
    public var commenttype: String? // "comment"
    public var added: Date?
    public var modified: Date?
    public var tsunix: Int64?
    public var userid: String?
    public var user: User?
    public var body: String?
    public var originalbody: String?
    public var hashtags: [String]?
    public var shadowban: Bool?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var edited: Bool?
    public var censored: Bool?
    public var deleted: Bool?
```

(continues on next page)

(continued from previous page)

```

public var parentid: String?
public var hierarchy: [String]?
public var reactions: [Reaction]?
public var likecount: Int64?
public var replycount: Int64?
public var votecount: Int64?
public var votescore: Int64?
public var votes: [Vote]?
public var moderation: String? // "approved", "pending", "rejected"
public var active: Bool?
public var reports: [Report]?

/// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.ReportComment(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1",
    userid: "yBommvwYYNBrxPwY",
    reporttype: ReportType.abuse
)

// Perform operation
commentClient.reportComment(request) { (code: Int?, message: String?, kind: String?,
↳ response: Comment?) in
    // ... Resolve `response` from here
}

```

## 15.16 Update Comment

```

func updateComment(_ request: CommentRequest.UpdateComment, completionHandler: @escaping
↳ Completion<Comment>)

```

UPDATES the contents of an existing comment

**Parameters**

- **conversationid** : (required) The ID of the comment conversation.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.
- **userid** : (required) The application specific user ID of the comment to be updated. This must be the owner of the comment or moderator / admin.
- **body** : (required) The new body contents of the comment.

The comment will be flagged to indicate that it has been modified.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.UpdateComment`

```

public class UpdateComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    public let userid: String // REQUIRED
    public let body: String // REQUIRED
    /// ...
}

```

### Response Model: Comment

```

open class Comment: Codable {
    public var kind: String? // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
    public var commenttype: String? // "comment"
    public var added: Date?
    public var modified: Date?
    public var tsunix: Int64?
    public var userid: String?
    public var user: User?
    public var body: String?
    public var originalbody: String?
    public var hashtags: [String]?
    public var shadowban: Bool?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var edited: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var parentid: String?
    public var hierarchy: [String]?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var replycount: Int64?
    public var votecount: Int64?
    public var votescore: Int64?
    public var votes: [Vote]?
    public var moderation: String? // "approved", "pending", "rejected"
    public var active: Bool?
    public var reports: [Report]?

    /// ...
}

```

### Example

```
let commentClient = CommentClient(config: config)
```

(continues on next page)

(continued from previous page)

```

let request = CommentRequest.UpdateComment(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1",
    userid: "yBommvwYYNBrxPwY",
    body: "Updated Comment!?!")
)

// Perform operation
commentClient.updateComment(request) { (code: Int?, message: String?, kind: String?,
↳ response: Comment?) in
    // ... Resolve `response` from here
}

```

## 15.17 Flag Comment As Deleted

```

func flagCommentLogicallyDeleted(_ request: CommentRequest.FlagCommentLogicallyDeleted,
↳ completionHandler: @escaping Completion<DeleteCommentResponse>)

```

Set Deleted (LOGICAL DELETE)

- The comment 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.

### Parameters

- **conversationid** : (required) The ID of the comment conversation.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.
- **userid** : (required) This is the application specific user ID of the user deleting the comment. Must be the owner of the comment or authorized moderator.
- **deleted** : (required) Set to true or false to flag the comment as deleted. If a comment is deleted, then it will have the deleted field set to true, in which case the contents of the comment should not be shown and the body of the comment will not be returned by the API by default. If a previously deleted comment is undeleted, the flag for deleted is set to false and the original comment body is returned.
- **permanentifnoreplies** : (optional) If this optional parameter is set to “true”, then if this comment has no replies it will be permanently deleted instead of logically deleted. If a permanent delete is performed, the result will include the field “permanentdelete=true”. If you want to mark a comment as deleted, and replies are still visible, use “true” for the logical delete value. If you want to permanently delete the comment and all of its replies, pass false.

**Warning** This method requires authentication

**Request Model:** CommentRequest.FlagCommentLogicallyDeleted

```

public class FlagCommentLogicallyDeleted: ParametersBase</*...*/> {
    /// ...
}

```

(continues on next page)

(continued from previous page)

```

public let conversationid: String // REQUIRED
public let commentid: String // REQUIRED
public let userid: String // REQUIRED
public let deleted: Bool // REQUIRED
public var permanentifnoreplies: Bool?
/// ...
}

```

**Response Model: DeleteCommentResponse**

```

open class DeleteCommentResponse: Codable {
    public var kind: String?
    public var permanentdelete: Bool?
    public var comment: Comment?

    /// ...
}

```

**Example**

```

let commentClient = CommentClient(config: config)
let request = CommentRequest.FlagCommentLogicallyDeleted(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1",
    userid: "yBommvwYYNBrxPwY",
    deleted: true,
    permanentifnoreplies: false
)

// Perform operation
commentClient.flagCommentLogicallyDeleted(request) { (code: Int?, message: String?,
↳ kind: String?, response: DeleteCommentResponse?) in
    // ... Resolve `response` from here
}

```

## 15.18 Delete Comment (permanent)

```

func permanentlyDeleteComment(_ request: CommentRequest.PermanentlyDeleteComment,
↳ completionHandler: @escaping Completion<DeleteCommentResponse>)

```

DELETES a comment and all replies to that comment

**Parameters**

- **conversationid** : (required) The ID of the comment conversation.
- **commentid** : (required) The unique ID of the comment, URL ENCODED.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.PermanentlyDeleteComment`

```
public class PermanentlyDeleteComment: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    public let commentid: String // REQUIRED
    /// ...
}
```

#### Response Model: DeleteCommentResponse

```
open class DeleteCommentResponse: Codable {
    public var kind: String?
    public var permanentdelete: Bool?
    public var comment: Comment?

    /// ...
}
```

#### Example

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.PermanentlyDeleteComment(
    conversationid: "demo-conversation-id1",
    commentid: "root-comment-id1"
)

// Perform operation
commentClient.permanentlyDeleteComment(request) { (code: Int?, message: String?, kind: String?, response: DeleteCommentResponse?) in
    // ... Resolve `response` from here
}
```

## 15.19 Delete Conversation

```
func deleteConversation(_ request: CommentRequest.DeleteConversation, completionHandler: @escaping Completion<DeleteConversationResponse>)
```

DELETES a Conversation, all Comments and Replies

- CANNOT BE UNDONE. This deletes all history of a conversation including all comments and replies within it.

#### Parameters

- **conversationid** : (required) The ID of the comment conversation.

**Warning** This method requires authentication

#### Request Model: CommentRequest.DeleteConversation

```
public class DeleteConversation: ParametersBase</*...*/> {
    /// ...
    public let conversationid: String // REQUIRED
    /// ...
}
```

**Response Model: DeleteConversationResponse**

```
open class DeleteConversationResponse: Codable {
    public var kind: String?
    public var permanentdelete: Bool?
    public var comment: Comment?

    /// ...
}
```

**Example**

```
let commentClient = CommentClient(config: config)
let request = CommentRequest.DeleteConversation(
    conversationid: "demo-conversation-id1"
)

// Perform operation
commentClient.deleteConversation(request) { (code: Int?, message: String?, kind: String?,
↳ response: DeleteConversationResponse?) in
    // ... Resolve `response` from here
}
```

## 15.20 List Comments in Moderation Queue

```
func listCommentsInModerationQueue(_ request: CommentModerationRequest.
↳ ListCommentsInModerationQueue, completionHandler: @escaping Completion
↳ <ListCommentsResponse>)
```

List all the comments in the moderation queue

**Parameters**

- **limit:** (optional) Defaults to 200. This limits how many messages to return from the queue
- **cursor:** (optional) Provide cursor value to get the next page of results.
- **conversationid:** (optional) Provide the ConversationID for a room to filter for only the queued events for a specific room
- **filterHandle:** (optional) Filters using exact match for a handle of a user
- **filterKeyword:** (optional) Filters using substring search for your string
- **filterModerationState:** (optional) Filters for comments in the specified moderation state.
  - *approved:* Moderator approved the comment
  - *rejected:* Moderator rejected the comment
  - *pending:* A new comment was posted to a premoderation room, and is pending review, but was never reported as abuse
  - *flagged:* Enough users reported the comment that it is in the flagged state and sent to moderation queue

**Warning** This method requires authentication

**Request Model:** `CommentRequest.ListCommentsInModerationQueue`



```
public class ListCommentsInModerationQueue: ParametersBase</*...*/> {
    /// ...
    public var limit: Int?
    public var cursor: String?
    public var conversationid: String?
    public var filterHandle: String?
    public var filterKeyword: String?
    public var filterModerationState: CommentModerationState?
    /// ...
}
```

#### Response Model: ListCommentsResponse

```
open class ListCommentsResponse: Codable {
    public var kind: String?
    public var permanentdelete: Bool?
    public var comment: Comment?

    /// ...
}
```

#### Example

```
let commentClient = CommentClient(config: config)
let request = CommentModerationRequest.ListCommentsInModerationQueue()
// You may provide optional parameters as shown below:
// request.limit = 25
// request.cursor = "63bd442ccfce070c7825639a"
// request.conversationid = "demo-conversation-id1"
// request.filterHandle = nil
// request.filterKeyword = "foul"
// request.filterModerationState = CommentModerationState.pending

// Perform operation
commentClient.listCommentsInModerationQueue(request) { (code: Int?, message: String?,
↳kind: String?, response: ListCommentsResponse?) in
    // ... Resolve `response` from here
}
```

## 15.21 Approve/Reject Message in Queue

```
func approveMessageInQueue(_ request: CommentModerationRequest.ApproveRejectComment,
↳completionHandler: @escaping Completion<Comment>)
```

APPROVES/REJECTS a message in the moderation queue.

If PRE-MODERATION is enabled for a conversation, then all messages go to the queue before they can appear in the conversation. For each incoming message, a webhook will be fired, if one is configured.

If the conversation is set to use POST-MODERATION, messages will only be sent to the moderation queue if they are reported.

#### Parameters

- **commentid** : (required) The unique ID of the comment, URL ENCODED.
- **approve** : (required) Pass true to approve the comment or false to reject the comment.

**Warning** This method requires authentication

**Request Model:** `CommentRequest.ApproveRejectComment`

```
public class ApproveRejectComment: ParametersBase</*...*/> {
    /// ...
    public let commentid: String    // REQUIRED
    public let approve: Bool        // REQUIRED
    /// ...
}
```

**Response Model:** `Comment`

```
open class Comment: Codable {
    public var kind: String?    // "comment.comment"
    public var id: String?
    public var appid: String?
    public var conversationid: String?
    public var commenttype: String? // "comment"
    public var added: Date?
    public var modified: Date?
    public var tsunix: Int64?
    public var userid: String?
    public var user: User?
    public var body: String?
    public var originalbody: String?
    public var hashtags: [String]?
    public var shadowban: Bool?
    public var customtype: String?
    public var customid: String?
    public var custompayload: String?
    public var customtags: [String]?
    public var customfield1: String?
    public var customfield2: String?
    public var edited: Bool?
    public var censored: Bool?
    public var deleted: Bool?
    public var parentid: String?
    public var hierarchy: [String]?
    public var reactions: [Reaction]?
    public var likecount: Int64?
    public var replycount: Int64?
    public var votecount: Int64?
    public var votescore: Int64?
    public var votes: [Vote]?
    public var moderation: String? // "approved", "pending", "rejected"
    public var active: Bool?
    public var reports: [Report]?

    /// ...
}
```

**Example**

```
let commentClient = CommentClient(config: config)
let request = CommentModerationRequest.ApproveRejectComment(
    commentid: "root-comment-id1",
    approve: true
)

// Perform operation
commentClient.approveMessageInQueue(request) { (code: Int?, message: String?, kind: String?, response: Comment?) in
    // ... Resolve `response` from here
}
```



## **COPYRIGHT & LICENSE**

Copyright (c) 2023 Sportstalk 24/7