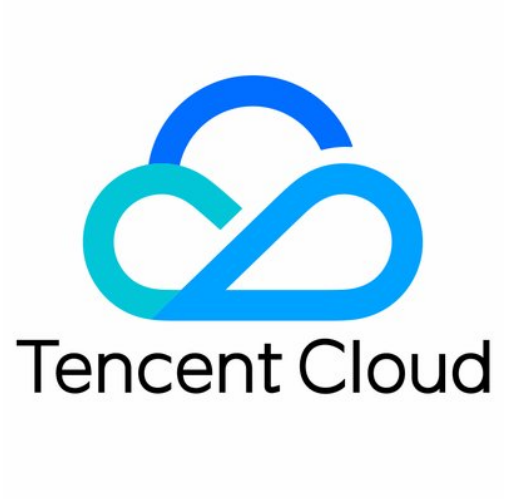
Chat

No UI Integration

Product Documentation



©2013-2025 Tencent Cloud International Pte. Ltd.

Copyright Notice

©2013-2025 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice

Tencent Cloud

All trademarks associated with Tencent Cloud and its services are owned by the Tencent corporate group, including its parent, subsidiaries and affiliated companies, as the case may be. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

- No UI Integration
  - Quick Start
    - JavaScript
    - React Native
  - SDK Integration
    - Android
    - iOS
    - JavaScript
    - React Native
    - Flutter
    - Windows
    - Mac
    - Unity
    - Unreal Engine
  - Initialization
    - Android&iOS&Windows&Mac
    - JavaScript
    - Flutter
    - Unity
    - React Native
  - Login and Logout
    - Android&iOS&Windows&Mac
    - JavaScript
    - Flutter
    - Unity
    - React Native
  - Message
    - Message Overview
      - Android&iOS&Windows&Mac
      - JavaScript
      - Flutter
      - Unity
      - React Native
    - Sending Message
      - Android&iOS&Windows&Mac

©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript
Flutter
Unity
React Native
Receiving Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Historical Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Forwarding Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Modifying Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Message Inserting
Android&iOS&Windows&Mac
Flutter
React Native
Deleting Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native

©2013-2025 Tencent Cloud International Pte. Ltd.

Clearing Messages
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Recalling Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Online Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Read Receipt
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Querying Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Group @ Message
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Targeted Group Message
Android&iOS&Windows&Mac

©2013-2025 Tencent Cloud International Pte. Ltd.

- JavaScript
- Flutter
- Unity
- React Native
- Notification Muting
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Message Extension
  - Android&iOS&Windows&Mac
  - JavaScript
  - Unity
  - Flutter
  - React Native
- Message Reaction
  - Android&iOS&Windows&Mac
  - JavaScript
  - React Native
- Message Translation
  - Android&iOS&Windows&Mac
  - JavaScript
  - React Native
- Message Pinning
  - Android&iOS&Windows&macOS

Conversation

- Conversation Overview
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Conversation List
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native
Getting Conversation
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Conversation Unread Count
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Pinning Conversation to the Top
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Deleting Conversation
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Conversation Draft
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Conversation Mark
Android&iOS&Windows&Mac
JavaScript
Flutter
Unity
React Native
Conversation Group

©2013-2025 Tencent Cloud International Pte. Ltd.

- Android&iOS&Windows&Mac
- JavaScript
- Flutter
- Unity
- React Native

Group

- Group Overview
- Group Management
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Group Profile
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Group Member Management
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Group Member Profile
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Custom Group Attribute
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Group Counter

©2013-2025 Tencent Cloud International Pte. Ltd.

- Android&iOS&Windows&Mac
- JavaScript
- React Native
- Unity

Community Topic

- Community Management
  - Android, iOS, Windows, and macOS
  - JavaScript
  - Flutter
  - React Native
- Permission Group
  - Android&iOS&Windows&Mac

User Profile and Relationship Chain

- User Profile
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- User Status
  - Android&iOS&Windows&Mac
  - JavaScript
  - Unity
  - React Native
- Friend Management
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Friend Group
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - Unity
  - React Native
- Blocklist
  - Android&iOS&Windows&Mac

©2013-2025 Tencent Cloud International Pte. Ltd.

- JavaScript
- Flutter
- Unity
- React Native
- Follow Feature
  - Android&iOS&Windows&Mac
  - JavaScript
  - Flutter
  - React Native
- Offline Push
  - Android
  - iOS
- Cloud Search
  - Cloud Search Messages
    - Android&iOS&Windows&Mac
    - JavaScript
    - React Native
  - Cloud Search Users
    - Android&iOS&Windows&Mac
    - JavaScript
    - React Native
  - Cloud Search Groups
    - Android&iOS&Windows&Mac
    - JavaScript
    - React Native
  - Cloud Search Group Members
    - Android&iOS&Windows&Mac
    - JavaScript
    - React Native
- Local Search
  - Searching for Message
    - Android&iOS&Windows&Mac
    - Flutter
    - Unity
  - Searching for Friend
    - Android&iOS&Windows&Mac
    - Flutter
    - Unity

©2013-2025 Tencent Cloud International Pte. Ltd.

- Searching Group
  - Android&iOS&Windows&Mac
  - Flutter
  - Unity
- Searching for Group Member
  - Android&iOS&Windows&Mac
  - Flutter
  - Unity

Signaling

- Signaling Management
  - Android
  - JavaScript
  - iOS
  - Flutter
  - React Native
  - Windows

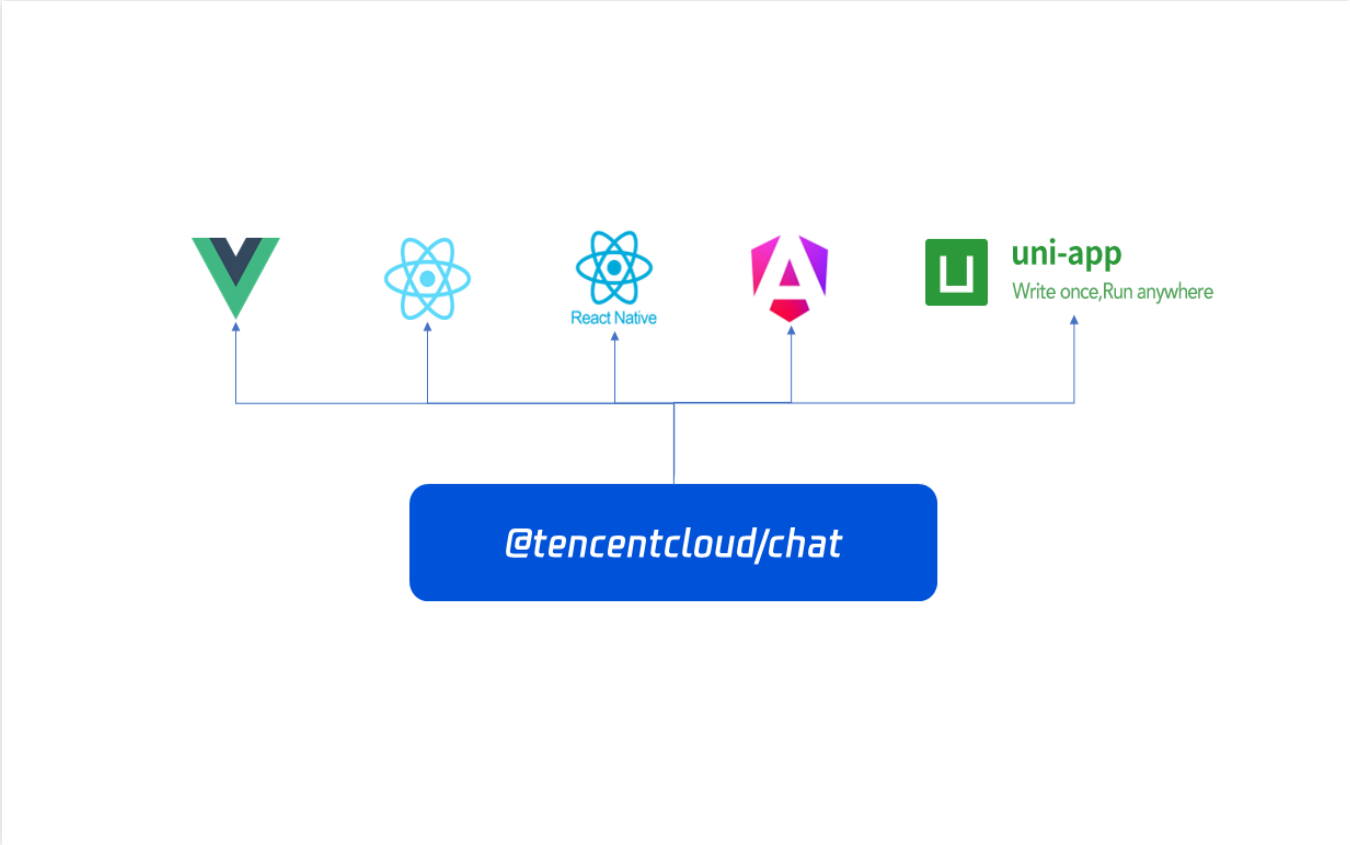
Official Channel Management

- Android&iOS&Windows&Mac

No UI Integration
Quick Start
JavaScript

Last updated：2024-12-13 11:18:11

The JavaScript SDK is an npm package, which is designed for seamless integration of chat functionality into your application. It provides an easy way to interact with chat APIs, allowing you to perform actions such as sending messages, creating groups, pinning conversations, and updating personal avatars. Written in vanilla JavaScript, it is framework-agnostic, meaning it can be used with any frontend framework like Vue, React, React Native, uni-app, Angular, and more.



Prior to reviewing the Chat API documentation, we suggest taking a look at the tutorials and sample apps available. This guide provides a quick introduction to the TencentCloudChat API, enabling you to quickly grasp its functionality. The API is highly flexible and empowers you to create various types of chat or messaging applications.

Initialize

Let's get started by initializing the chat instance and listening on events.

```
import TencentCloudChat from '@tencentcloud/chat';
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
import TIMUploadPlugin from 'tim-upload-plugin';

let options = {
  SDKAppID: 0 // Replace 0 with the SDKAppID of your Chat application when connecti
};
// Create an SDK instance.
// The `TencentCloudChat.create()` method returns the same instance for the same `S
// The SDK instance is usually represented by chat.
let chat = TencentCloudChat.create(options);

// Set the SDK log level.
// 0: Common level. You are advised to use this level during access as it covers mo
// 1: Release level. You are advised to use this level for key information in a pro
chat.setLogLevel(0);
// chat.setLogLevel(1);

// Register the Tencent Cloud Chat upload plugin.
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});

let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array to store Messages - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

And then, Log in to Chat.

```
await chat.login({userID: 'your userID', userSig: 'your userSig'});
```

Messages

Let’s continue by sending your first message to "userB"(Assuming "userB" has already logged into the chat).

©2013-2025 Tencent Cloud International Pte. Ltd.




```
let message = chat.createTextMessage({
  to: 'userB',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    text: 'Hello World!'
  },
});

await chat.sendMessage(message);
```

Conversations

If the chat with "userB" is important to you and you need to place it at the top of the conversation list, you can use `pinConversation` .

```
await chat.pinConversation({ conversationID: 'C2CuserB', isPinned: true });
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Profiles

If you want to change your avatar, you can use `updateMyProfile` .

```
await chat.updateMyProfile({
  avatar: 'http(s)://url/to/image.jpg',
});
```

Groups

If you want to create a group chat, for example, to discuss the sales plan for the next quarter with subordinates and colleagues, you can use `createGroup` .

```
await chat.createGroup({
  type: TencentCloudChat.TYPES.GRP_WORK,
  name: 'Sales Plan For The Next Quarter',
  memberList: [{
    userID: 'lindal',
  }, {
    userID: 'denny',
  }] // If `memberList` is specified, `userID` must also be specified.
});
```

Following & Followers

©2013-2025 Tencent Cloud International Pte. Ltd.



Now live streaming and short videos are very popular. If you want to follow a certain celebrity, you can use `followUser`.

```
await chat.followUser(['celebrity1', 'celebrity2', 'celebrity3']);

// Get my follower list
await chat.getMyFollowersList();
```

Want to watch the live streaming and post some interesting comments? You can use `joinGroup` to join a live group and then use `createTextMessage` to create a message and then use `sendMessage` to post it.

```
await chat.joinGroup({ groupID: 'group1' });

let message = chat.createTextMessage({
  to: 'group1',
  conversationType: TencentCloudChat.TYPES.CONV_GROUP,
  payload: {
    text: 'AMAZING!!!'
  },
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
});

await chat.sendMessage(message);
```

Conclusion

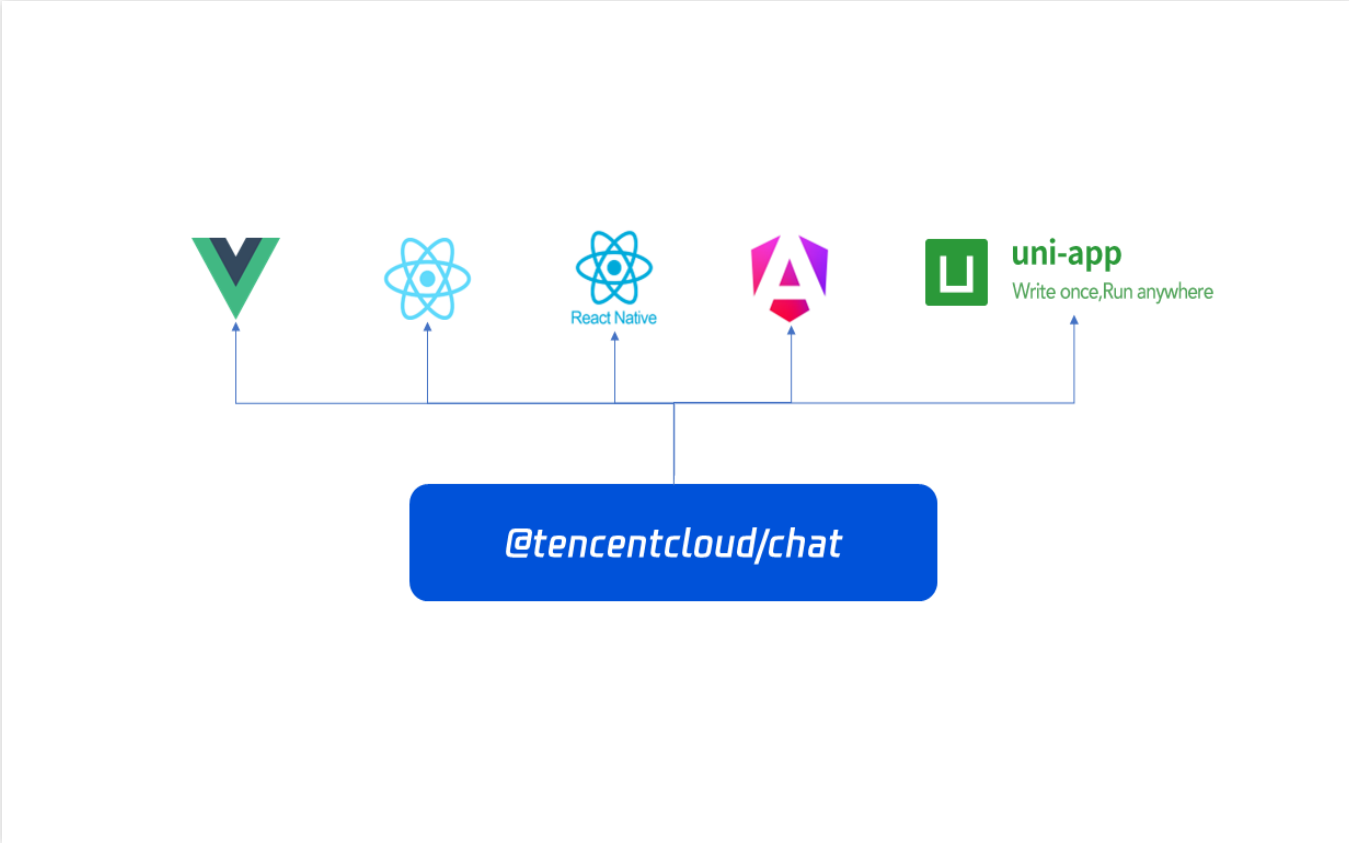
Since you now have a grasp of the fundamental components required for a fully operational chat integration, let's proceed to the subsequent sections of the documentation. In these parts, we will explore the specifics of each API endpoint in greater depth.

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

Last updated：2025-01-07 16:23:55

The React Native SDK is an npm package, which is designed for seamless integration of chat functionality into your application. It provides an easy way to interact with chat APIs, allowing you to perform actions such as sending messages, creating groups, pinning conversations, and updating personal avatars. Written in vanilla JavaScript, it is framework-agnostic, meaning it can be used with any frontend framework like Vue, React, React Native, uni-app, Angular, and more.



Prior to reviewing the Chat API documentation, we suggest taking a look at the tutorials and sample apps available. This guide provides a quick introduction to the TencentCloudChat API, enabling you to quickly grasp its functionality. The API is highly flexible and empowers you to create various types of chat or messaging applications.

Initialize

Let's get started by initializing the chat instance and listening on events.

```
import TencentCloudChat from '@tencentcloud/chat';
import TIMUploadPlugin from 'tim-upload-plugin';
import NetInfo from '@react-native-community/netinfo';

let options = {
  SDKAppID: 0 // Replace 0 with the SDKAppID of your Chat application when connecti
```

```
};
// Create an SDK instance.
// The `TencentCloudChat.create()` method returns the same instance for the same `S
// The SDK instance is usually represented by chat.
let chat = TencentCloudChat.create(options);

// Set the SDK log level.
// 0: Common level. You are advised to use this level during access as it covers mo
// 1: Release level. You are advised to use this level for key information in a pro
chat.setLogLevel(0);
// chat.setLogLevel(1);

// Register the Tencent Cloud Chat upload plugin.
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});
// Register the network monitoring plugin
chat.registerPlugin({'chat-network-monitor': NetInfo});

let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array to store Messages - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

And then, Log in to Chat.

```
await chat.login({userID: 'your userID', userSig: 'your userSig'});
```

Messages

Let’s continue by sending your first message to "userB"(Assuming "userB" has already logged into the chat).

©2013-2025 Tencent Cloud International Pte. Ltd.




```
let message = chat.createTextMessage({
  to: 'userB',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    text: 'Hello World!'
  },
});

await chat.sendMessage(message);
```

Conversations

If the chat with "userB" is important to you and you need to place it at the top of the conversation list, you can use `pinConversation` .

```
await chat.pinConversation({ conversationID: 'C2CuserB', isPinned: true });
```

Profiles

If you want to change your avatar, you can use `updateMyProfile` .

```
await chat.updateMyProfile({
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  avatar: 'http(s)://url/to/image.jpg',
});
```

Groups

If you want to create a group chat, for example, to discuss the sales plan for the next quarter with subordinates and colleagues, you can use `createGroup` .

```
await chat.createGroup({
  type: TencentCloudChat.TYPES.GRP_WORK,
  name: 'Sales Plan For The Next Quarter',
  memberList: [{
    userID: 'lindal',
  }, {
    userID: 'denny',
  }] // If `memberList` is specified, `userID` must also be specified.
});
```

Following & Followers



Now live streaming and short videos are very popular. If you want to follow a certain celebrity, you can use `followUser` .

```
await chat.followUser(['celebrity1', 'celebrity2', 'celebrity3']);

// Get my follower list
await chat.getMyFollowersList();
```

Want to watch the live streaming and post some interesting comments? You can use `joinGroup` to join a live group and then use `createTextMessage` to create a message and then use `sendMessage` to post it.

```
await chat.joinGroup({ groupID: 'group1' });

let message = chat.createTextMessage({
  to: 'group1',
  conversationType: TencentCloudChat.TYPES.CONV_GROUP,
  payload: {
    text: 'AMAZING!!!'
  },
});

await chat.sendMessage(message);
```

Conclusion

Since you now have a grasp of the fundamental components required for a fully operational chat integration, let's proceed to the subsequent sections of the documentation. In these parts, we will explore the specifics of each API endpoint in greater depth.

©2013-2025 Tencent Cloud International Pte. Ltd.

SDK Integration
Android

Last updated：2025-03-26 15:53:54

This document describes how to quickly integrate the Chat SDK into your Android project.

Environment Requirements

JDK 1.6.
Android 4.1 (SDK API level 16) or above

Integrating SDK (AAR)

You can use Gradle to automatically load the AAR file or manually download the AAR file and import it into your project.

Method 1: Automatic Loading (AAR)

You can configure Gradle to automatically download the latest Chat SDK that has been released in the Maven Central library.
Use Android Studio to open your project and modify the `app/build.gradle` file in three simple steps to integrate the SDK to your project, as shown below:

Step 1. Add SDK Dependencies

1. Find `build.gradle` of the app and add `mavenCentral()` dependencies to `repositories` .

```
repositories {
    google()
    jcenter()
    // Add the `mavenCentral` repository.
    mavenCentral()
}
```

2. Add the Chat SDK dependencies to `dependencies` .

```
dependencies {
    // Add the SDK and use the latest version number as recommended
    api 'com.tencent.imsdk:imsdk-plus:Version number'
```

```
    // If you need to add the Quic plugin, please uncomment the next line
(Note: the plugin version number must match the Chat SDK version number)
    // api "com.tencent.imsdk:timquic-plugin:Version number"
}
```

Replace `Version number` with the actual version number of the SDK. We recommend you use the latest version.Take the version number 5.4.666 as an example:

```
dependencies {
    api 'com.tencent.imsdk:imsdk-plus:5.4.666'
}
```

Note

Quic plugin provides axp-quic multiplexing protocol, which has better weak network resistance and can still provide services even when the network packet loss rate reaches 70%. This plugin is only available on the Chat Pro edition、Pro Plus edition、Enterprise edition, purchase the Pro edition、Pro Plus edition、Enterprise edition to use it. To ensure proper functionality, please update the client SDK to version 7.7.5282 or higer.

Step 2. Specify the App Architecture

In `defaultConfig`, specify the CPU architecture used by the app (armeabi-v7a, arm64-v8a, x86, and x86_64 are supported starting from Chat SDK v4.3.118).

```
defaultConfig {
    ndk {
        abiFilters "arm64-v8a"
    }
}
```

Step 3. Sync the SDK

Click the `Sync` icon. If the connection to JCenter is normal, the SDK will be automatically downloaded and integrated to your project.

Method 2: Manual Download (AAR)

If JCenter cannot be accessed, you can manually download the SDK and integrate it into your project:

Step 1. Download SDK

Download the latest version of the Chat SDK from GitHub.

Step 2. Copy SDK to the Project Directory

©2013-2025 Tencent Cloud International Pte. Ltd.

Copy the downloaded AAR file to the /**libs** directory of the project.

Step 3. Specify, Compile, and Run

In the `defaultConfig` of `app/build.gradle`, specify the CPU architecture used by the app (armeabi-v7a, arm64-v8a, x86, and x86_64 are supported starting from SDK v4.3.118).

```
defaultConfig {
    ndk {
        abiFilters "arm64-v8a"
    }
}
```

Note：

If you need to add a Quic plugin, please refer to the previous steps and manually download the integrated Quic plugin.

Integrating SDK

If you do not want to integrate the AAR library, you can integrate the Chat SDK by importing the JAR and SO libraries.

Step 1. Download and Decompress SDK

Download the latest version of the AAR file from GitHub and decompress it. The extracted folder contains a JAR file and an SO subfolder. Rename **classes.jar** to **imsdk.jar**.

Step 2. Copy SDK to the Project Directory

Copy the renamed JAR file and SO files of different architectures to the default loading directories of Android Studio.

Step 3. Specify, Compile, and Run

In the `defaultConfig` of `app/build.gradle`, specify the CPU architecture used by the app (armeabi-v7a, arm64-v8a, x86, and x86_64 are supported starting from Chat SDK v4.3.118).

©2013-2025 Tencent Cloud International Pte. Ltd.

```
defaultConfig {
    ndk {
        abiFilters "arm64-v8a"
    }
}
```

Configuring Permissions

To configure app permissions in `AndroidManifest.xml`, the Chat SDK requires the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

Configuring Obfuscation Rules

In the `proguard-rules.pro` file, add the Chat SDK classes to the "do not obfuscate" list.

```
-keep class com.tencent.imsdk.** { *; }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

iOS

Last updated：2025-03-26 15:53:54

This document describes how to quickly integrate the Chat SDK to your projects.

Environment Requirements

Xcode 9.0+.
iPhone or iPad on iOS 8.0 or above.
Your project has a valid developer signature.

Integrating SDK

You can either automatically integrate SDK using CocoaPods, or manually download the SDK and import it to your current project.

CocoaPods

1. Install CocoaPods

Enter the following command in the terminal window (you need to install the Ruby environment on your macOS in advance):

```
sudo gem install cocoapods
```

2. Create a Podfile

Go to the path where the project is located and run the following command. Then, a Podfile will appear under the project path.

```
pod init
```

3. Edit the Podfile

Add the dependency to your Podfile:

```
platform :ios, '8.0'
source 'https://github.com/CocoaPods/Specs.git'

target 'App' do
```

```
    # Add the SDK
    pod 'TXIMSDK_Plus_iOS'
    # pod 'TXIMSDK_Plus_iOS_XCFramework'
    # pod 'TXIMSDK_Plus_Swift_iOS_XCFramework'

    # If you need to add the Quic plugin, please uncomment the next line.
    # Note:
    # - This plugin must be used with the TXIMSDK_Plus_iOS or
TXIMSDK_Plus_iOS_XCFramework edition of the IM SDK, and the plugin version
number must match the IM SDK version number.
    # - For the TXIMSDK_Plus_Swift_iOS_XCFramework edition, there is no need to
add this plugin. If you need to use the Quic feature in this edition, please
contact us.
    # pod 'TXIMSDK_Plus_QuicPlugin'
end
```

4. Install the SDK or update the local repository.

Run the following command in the terminal window to update the local library file and install the Chat SDK:

```
pod install
```

Or, run the following command to update the local repository:

```
pod update
```

After the pod command is executed, an .xcworkspace project file integrated with the SDK will be generated. Double-click this file to open it. If the pod search fails, you are advised to update the local repo cache of the pod by running the following commands:

```
pod setup
pod repo update
rm ~/Library/Caches/CocoaPods/search_index.json
```

Note:

Quic plugin provides axp-quic multiplexing protocol, which has better weak network resistance and can still provide services even when the network packet loss rate reaches 70%. This plugin is only available on the Chat Pro edition、Pro Plus edition、Enterprise edition, purchase the Pro edition、Pro Plus edition、Enterprise edition to use it. To ensure proper functionality, please update the client SDK to version 7.7.5282 or higer.

If you need to use the Quic feature in the Swift edition of the SDK, please contact us through the Telegram technical group.

Manual integration

1. Download the SDK

Download the latest SDK version from Github. `ImSDK_Plus.framework` are the core dynamic library files of the SDK.

2. Create a project

Create a project.

Enter a project name, for example, IMDemo.

3. Integrate the SDK

Add the dependency library: select Target for IMDemo. On the General panel, add the dependency library under Embedded Binaries and Linked Frameworks and Libraries, select ImSDK_Plus.framework.

Set link parameters: add `-ObjC` in Build Setting -> Other Linker Flags.

Note：

For manual integration, you need to change `ImSDK_Plus.framework` to `Embed&Sign` in Target -> General -> Frameworks -> Libraries and Embedded Content.

If you need to add a Quic plugin, please refer to the previous steps and manually download the integrated Quic plugin.

Import SDK

There are two ways to use the SDK in your project code.

Method 1

Choose Xcode -> Build Setting -> Header Search Paths, and set the SDK header file path. In files that require the SDK API, reference the corresponding header file.

```
#import "ImSDK_Plus.h"
```

Method 2

In files that require the SDK API, reference the corresponding header file.

```
#import <ImSDK_Plus/ImSDK_Plus.h>
```

FAQs

[Xcodeproj] Unknown Object Version (60). (RuntimeError)

©2013-2025 Tencent Cloud International Pte. Ltd.

When you use Xcode 15 to create a project to integrate TUIKit, entering `pod install` may result in this issue. The reason is that an earlier version of CocoaPods is used. The following two solutions are available:
Solution 1: Change the `Project Format` version of the Xcode project.

Solution 2: Upgrade the local version of CocoaPods. The upgrading method is not covered in this document. You can enter `pod --version` in the terminal to check the current version of Pods.

Xcode 15 Developer Sandbox Option Issue

Sandbox: bash(xxx) deny(1) file-write-create

When using Xcode 15 to create a project, you may encounter compilation failures due to this option. It is recommended that you disable this option.

Xcode 16 does not support enabling bitcode for Frameworks

Solution 1: Upgrade the SDK

If you are using an old version SDK with Bitcode (e.g., TXIMSDK_iOS), it is recommended to follow this document and upgrade the SDK to TXIMSDK_Plus_iOS_XCFramework.

Solution 2: Modify the Podfile

Add the following configuration to the end of your Podfile and run pod install again.

```
post_install do |installer|
  bitcode_strip_path = 'xcrun --find bitcode_strip'.chop!
  def strip_bitcode_from_framework(bitcode_strip_path, framework_relative_path)
    framework_path = File.join(Dir.pwd, framework_relative_path)
    command = "#{bitcode_strip_path} #{framework_path} -r -o #{framework_path}"
    puts "Stripping bitcode: #{command}"
    system(command)
  end
  framework_paths = [
    "/Pods/TXIMSDK_iOS/ImSDK.framework/ImSDK",
  ]
  framework_paths.each do |framework_relative_path|
    strip_bitcode_from_framework(bitcode_strip_path, framework_relative_path)
   end
end
```

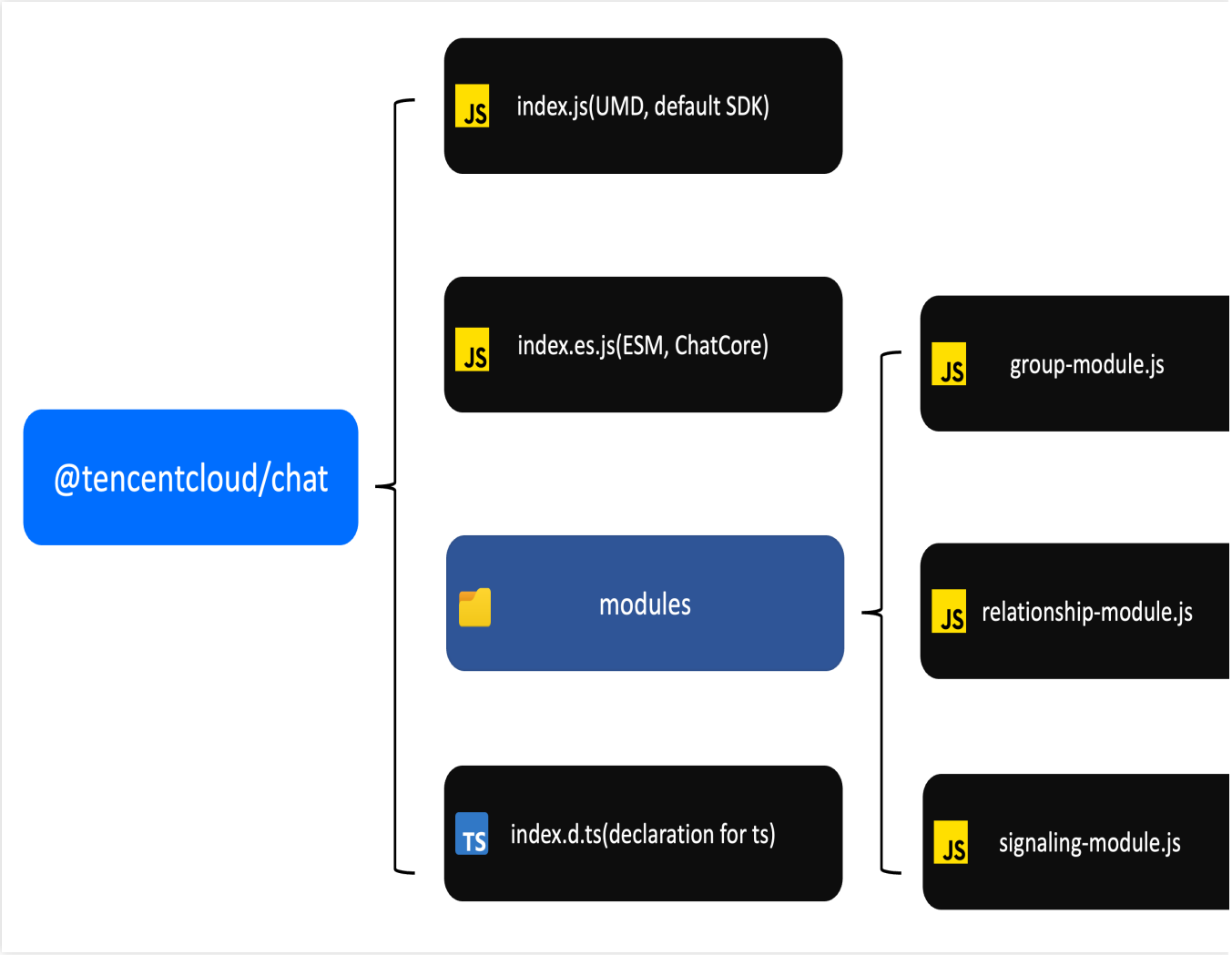
©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript

Last updated：2025-01-07 15:15:29

This document describes how to quickly integrate the Tencent Cloud Chat SDK into your web, mini program, uni-app and React Native projects.

File Structure of the Chat SDK



Integrating the SDK

You can integrate the Chat SDK into your web, mini program, and uni-app projects by using npm.
You can integrate the upload plugin tim-upload-plugin for faster and safer upload of rich media resources.

Integration via npm (Recommended)

Use npm to install the corresponding Chat SDK dependencies in your project.

```
npm install @tencentcloud/chat --save
// The Tencent Cloud Chat upload plugin is required to send messages such as images
npm install tim-upload-plugin --save
```

```
import TencentCloudChat from '@tencentcloud/chat';
import TIMUploadPlugin from 'tim-upload-plugin';

let options = {
  SDKAppID: 0 // Replace 0 with the SDKAppID of your Chat application when connecti
};
// Create an SDK instance.
// The `TencentCloudChat.create()` method returns the same instance for the same `S
// The SDK instance is usually represented by chat.
let chat = TencentCloudChat.create(options);

// Set the SDK log level.
// 0: Common level. You are advised to use this level during access as it covers mo
// 1: Release level. You are advised to use this level for key information in a pro
chat.setLogLevel(0);
// chat.setLogLevel(1);

// Register the Tencent Cloud Chat upload plugin.
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});
```

Set Log Level

Set the log level. Logs below the specified level will not be output.

API

```
chat.setLogLevel(level);
```

Name	Type	Description
level	Number	Log levels: 0 - Normal level, with a larger amount of logs. It is recommended for use during integration. 1 - Release level, the SDK outputs key information. It is recommended for use in production environments. 2 - Warning level, the SDK only outputs warning and error level logs. 3 - Error level, the SDK only outputs error level logs. 4 - No log level, the SDK will not print any logs.

©2013-2025 Tencent Cloud International Pte. Ltd.

Examples

```
chat.setLogLevel(0);
```

Register Plugins

Note：

1. To send messages such as images, voice, video, and files, you need to use the upload plugin `tim-upload-plugin` to upload files to Tencent Cloud Object Storage.
2. When using `tim-upload-plugin` in a React Native project, you need to upgrade `tim-upload-plugin` to version v1.3.1 or above. Also, to improve the user experience in poor network conditions or during network switching, you need to register the network status listening plugin `@react-native-community/netinfo`.
3. To send rich media messages in a React Native packaged App, you need to add permissions:

Add the following permissions to the `AndroidManifest.xml` in the Android project:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Add the following permissions to the `Info.plist` in the iOS project:

```
<key>NSCameraUsageDescription</key>
<string>We need access to your camera to take photos</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>We need access to your photo library to select photos</string>
<key>NSMicrophoneUsageDescription</key>
<string>We need access to your microphone to record audio</string>
```

API

```
chat.registerPlugin(options);
```

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
key	String	Plugin name.
value	Class	Plugin class.

Examples

```
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// Register the network status listening plugin in a React Native project.
npm install @react-native-community/netinfo --save

import NetInfo from "@react-native-community/netinfo";
chat.registerPlugin({'chat-network-monitor': NetInfo});
```

Next step, initialize Chat SDK.

Relevant Resources

SDK Update Logs

SDK APIs

FAQs

Are there any open-source UI components that can be reused or redeveloped?

Tencent Cloud Chat provides open-source UIKits for all platforms that can be reused or redeveloped by developers. Find the reference documentation below:

Quick Start（React）

Quick Start（Vue）

React Native

Last updated：2025-01-07 15:15:29

This document describes how to quickly integrate the Tencent Cloud Chat SDK into your React Native project.

Note:

1. Integrating @tencentcloud/chat into your React Native project requires downloading version 3.4.3 or later.
2. Integrating tim-upload-plugin into your React Native project requires downloading version 1.4.0 or later.

Environment Requirements

Platform	Version
React Native	v0.75.0 or later.
Android	Android Studio 3.5 or later. The app requires Android 4.1 or later devices.
iOS	Xcode 11.0 or later. For testing with a real device, make sure that your project has a valid developer signature.
Node	Version 18.0 or later.

Configuring the development environment

If this is your first time developing a React Native project, refer to the React Native official steps set-up-your-environment to configure the development environment.
If you encounter environment problems when creating or compiling the project, you can run `npx react-native doctor` for environment diagnostics.

Create a project (can be skipped if you already have one)

```
npx react-native@latest init chatExample
```

`npm react-native` creates projects with Android gradle version 8.8. If your Android gradle version is lower than 8.8, please open Android Studio to sync the gradle version.

©2013-2025 Tencent Cloud International Pte. Ltd.

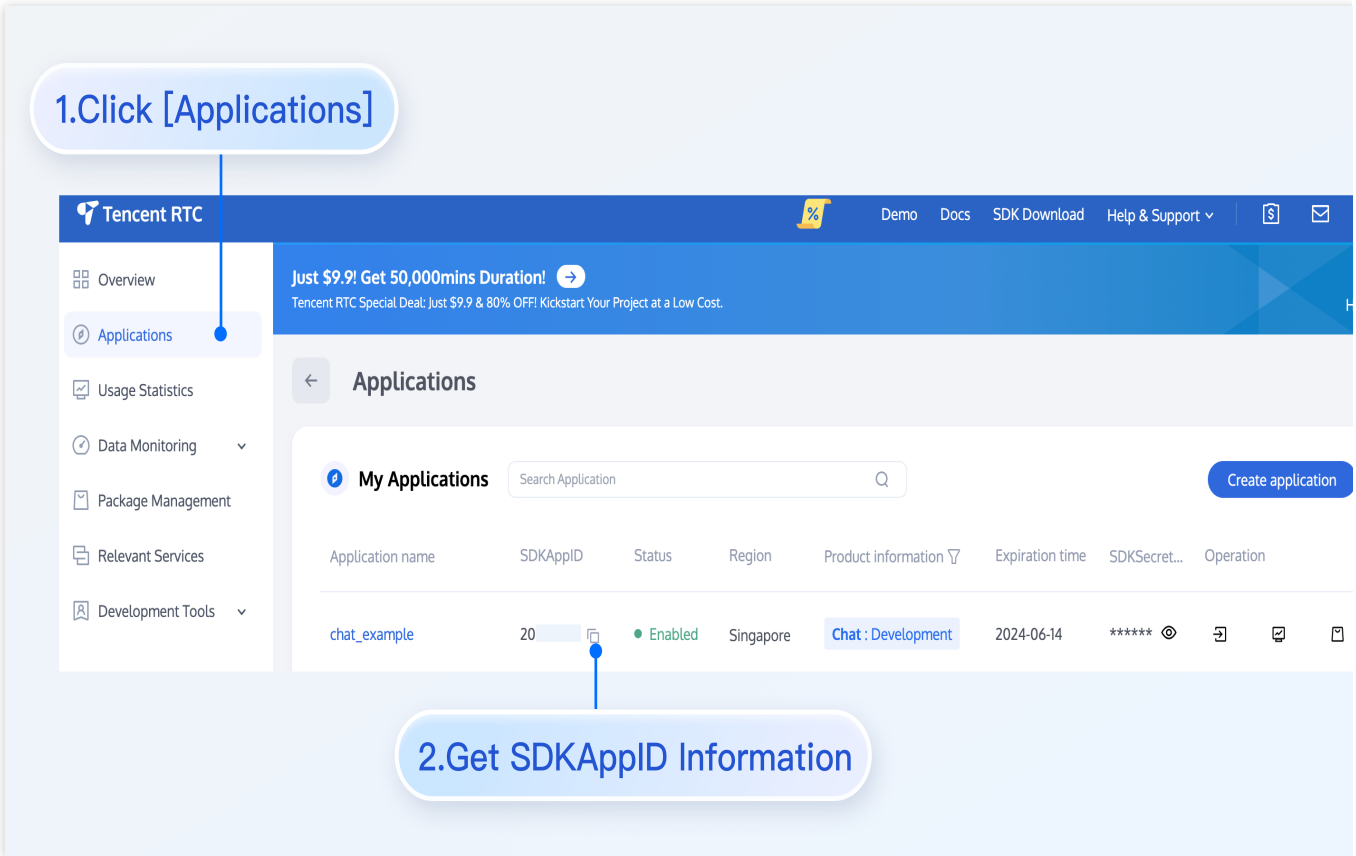
Integrate Chat SDK

Integrate the Chat SDK into your React Native project using npm.
You can integrate the upload plugin tim-upload-plugin for faster and safer upload of rich text message resources.
Enhance the user experience in a poor network environment or during network switching by integrating @react-native-community/netinfo.
Install the Chat SDK dependencies in your React Native project.

```
npm install @tencentcloud/chat tim-upload-plugin  @react-native-community/netinfo --save
```

Initialization

You must have the correct SDKAppID to proceed with the initialization.
SDKAppID is the unique identifier Tencent Cloud IM uses to distinguish customer accounts. We recommend applying for a new SDKAppID for each independent app. Messages between different SDKAppIDs are inherently isolated and cannot be interoperated.
You can view all SDKAppIDs in the Chat console. Click **Create application** to create a new SDKAppID.



Import the Chat SDK and initialize it in your project's App.tsx.

```
import TencentCloudChat from '@tencentcloud/chat';
import TIMUploadPlugin from 'tim-upload-plugin';
import NetInfo from '@react-native-community/netinfo';

let options = {
  SDKAppID: 0 // Replace 0 with the SDKAppID of your Chat application when connecti
};
// Create an SDK instance. The `TencentCloudChat.create()` method returns the same
let chat = TencentCloudChat.create(options); // The SDK instance is usually referre

chat.setLogLevel(0); // Normal level, with a lot of logs; it's recommended for inte

// Register the Tencent Cloud Instant Messaging rich media resource upload plugin
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});
// Register the network monitoring plugin
chat.registerPlugin({'chat-network-monitor': NetInfo});
```

Listening on events

SDK_READY

This is triggered when the SDK enters the ready state. After the accessing side listens to this event, it can call SDK APIs to send messages or use other SDK features.

```
let onSdkReady = function(event) {
  // After listening to the SDK ready event, you can make API calls
};
chat.on(TencentCloudChat.EVENT.SDK_READY, onSdkReady);
```

SDK_NOT_READY

This is triggered when the SDK enters the not-ready state. At this time, the accessing side will not be able to use features like sending messages via the SDK. To resume usage, the accessing side needs to call the login interface and drive the SDK to enter the ready state.

```
let onSdkNotReady = function(event) {
  // chat.login({userID: 'your userID', userSig: 'your userSig'});
};
chat.on(TencentCloudChat.EVENT.SDK_NOT_READY, onSdkNotReady);
```

MESSAGE_RECEIVED

©2013-2025 Tencent Cloud International Pte. Ltd.

When the SDK receives new messages from one-on-one chat, group chat, group prompts, or group system notifications, the accessing side can traverse event.data to obtain the message list data and render it to the UI.

```
let onMessageReceived = function(event) {
  // event.data - An array that stores `Message` objects - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

CONVERSATION_LIST_UPDATED

The conversation list is updated. event.data is an array containing Conversation objects.

```
let onConversationListUpdated = function(event) {
  console.log(event.data); // Array that stores Conversation instances
};
chat.on(TencentCloudChat.EVENT.CONVERSATION_LIST_UPDATED, onConversationListUpdated
```

Note:

If you need to learn more about SDK event notifications, please view the SDK event list.

Deinitialization

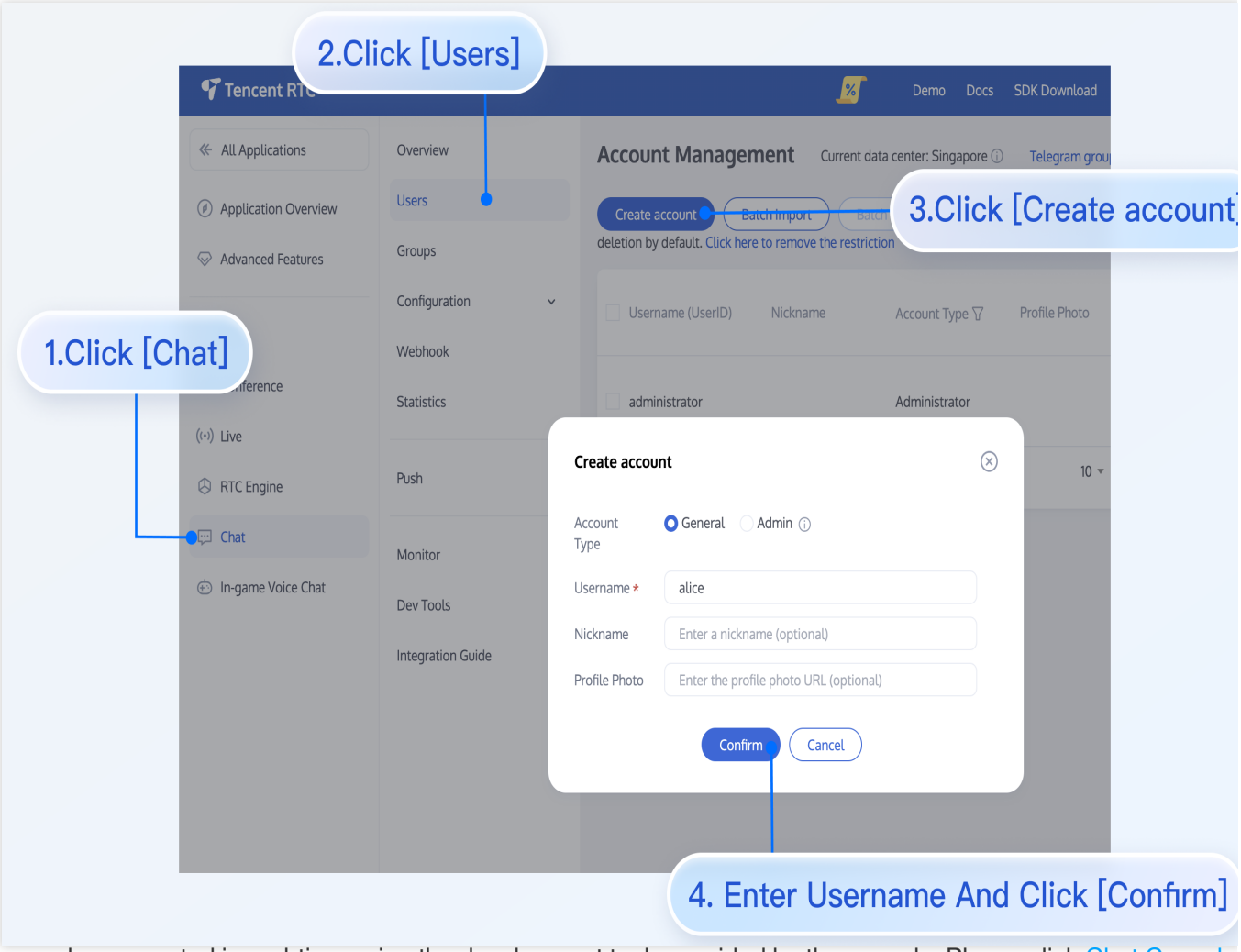
Terminate the SDK instance. The SDK will log out, disconnect the WebSocket persistent connection, and then release resources.

```
chat.destroy();
```

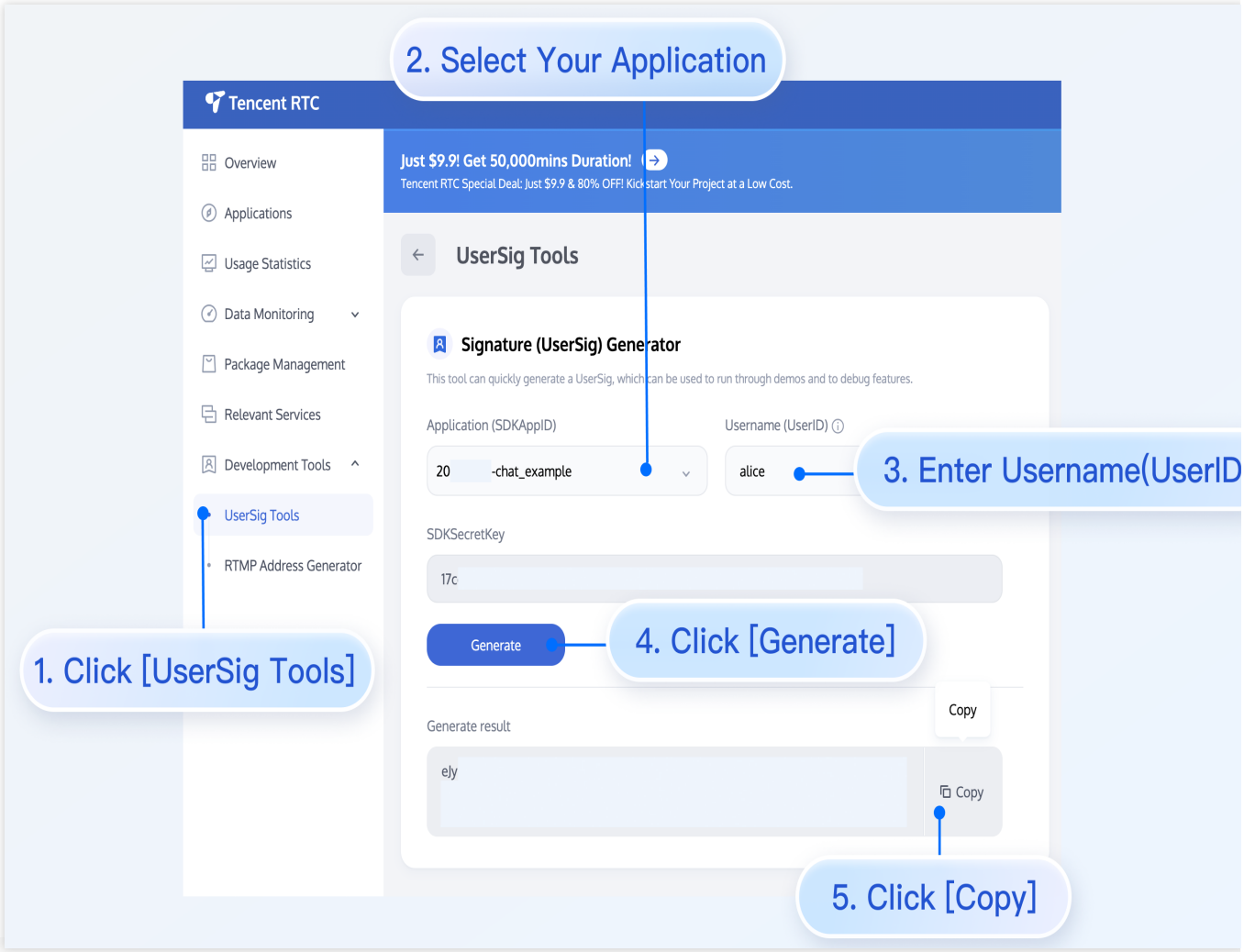
Login

Log in requires providing information such as userID and userSig. Please log in to the Chat console to obtain them.
userID
Click to enter the Application you created, and you will see the Chat Product entry in the left sidebar. Click to enter.
After entering the Chat Product subpage, click Users to enter the user management page.
Click Create account to open the account information form. If it is for a regular member, we recommend choosing the General type.
To enhance your experience with message sending and receiving features, we recommend creating two userIDs.

©2013-2025 Tencent Cloud International Pte. Ltd.



userSig can be generated in real-time using the development tools provided by the console. Please click Chat Console > Development Tools > UserSig Tools > Signature (UserSig) Generator.




```
let promise = chat.login({
  userID: 'your userID',
  userSig: 'your userSig',
});
promise.then(function(imResponse) {
  if (imResponse.data.repeatLogin === true) {
    // This indicates that the account is already logged in. This login attempt is
    console.log(imResponse.data.errorInfo);
  }
}).catch(function(imError) {
  // Information related to login failure
  console.warn('login error:', imError);
});
```

Compile and run the React Native application

To compile and run the project, you need to use a real device or an emulator.**A real device is recommended.** You can refer to the React Native official website running-on-device to connect a real device for debugging.

Android

iOS

1. Enable Developer Mode on your phone, and turn on the**USB Debugging** switch.
2. Connect your phone via USB. It's recommended to select the **Transfer files** option, **do not choose the Charging only option**.
3. After confirming the successful connection of your phone, execute `npm run android` to compile and run the project.

```
npm run android
```

1. Connect your mobile phone with a USB cable, and open the ios directory of the project using Xcode.
2. Configure the signing information according to the React Native official website running-on-device.
3. Go to the ios directory and install dependencies.

```
cd ios
pod install
```

4. Go back to the root directory and execute `npm run ios` to compile and run the project.

```
cd ../
npm run ios
```

Integrate third-party module

If you need to implement features like sending**Images, Videos, Files, Voice**, it is recommended to use the third-party modules provided below.

Select Images and Videos, Take Photo, Shoot Video, you need to integrate react-native-image-picker.

```
npm install react-native-image-picker --save
```

Select Image

Take Photo

Select Video

Record Video

```
import {launchImageLibrary} from 'react-native-image-picker';

// 1. Select an Image
launchImageLibrary({
```

```
  mediaType: 'photo',
  selectionLimit: 1,
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createImageMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Image
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message was successfully sent
    console.log(imResponse);
  }).catch(function(imError) {
    // The message failed to be sent
    console.warn('sendMessage error:', imError);
  });
});
```

```
import {launchCamera} from 'react-native-image-picker';
// 1. Take Photo
launchCamera({
  mediaType: 'photo',
  cameraType: 'back',
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createImageMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Image
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message was successfully sent
    console.log(imResponse);
  }).catch(function(imError) {
    // The message failed to be sent
    console.warn('sendMessage error:', imError);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  });
});
```

```
import {launchImageLibrary} from 'react-native-image-picker';

// 1. Select a Video
launchImageLibrary({
  mediaType: 'video',
  selectionLimit: 1,
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createVideoMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Video
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message was successfully sent
    console.log(imResponse);
  }).catch(function(imError) {
    // The message failed to be sent
    console.warn('sendMessage error:', imError);
  });
});
```

```
import {launchCamera} from 'react-native-image-picker';

// 1. Record Video
launchCamera({
  mediaType: 'video',
  cameraType: 'back',
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createVideoMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
```

```
  // 3. Send Video
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message was successfully sent
    console.log(imResponse);
  }).catch(function(imError) {
    // The message failed to be sent
    console.warn('sendMessage error:', imError);
  });
});
```

Select Files, you need to integrate react-native-document-picker.

```
npm install react-native-document-picker --save
```

```
import DocumentPicker from 'react-native-document-picker';

// 1. Select File
DocumentPicker.pick({
  type: [DocumentPicker.types.allFiles],
}).then((result) => {
 const file = result[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createFileMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send File
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message was successfully sent
    console.log(imResponse);
  }).catch(function(imError) {
    // The message failed to be sent
    console.warn('sendMessage error:', imError);
  });
});
```

Record voice, you need to integrate react-native-audio-recorder-player.

```
npm install react-native-audio-recorder-player --save
```

```
import AudioRecorderPlayer, {AVEncodingOption} from 'react-native-audio-recorder-pl
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// Record recording duration
let duration = 0;
// Record recording file path
let uri = '';

// 1. Start recording
const onStartRecord = async () => {
  await audioRecorderPlayer.startRecorder('test.aac',{
    VFormatIDKeyIOS: AVEncodingOption.aac,
  });
  audioRecorderPlayer.addRecordBackListener((e: any) => {
    duration = e.currentPosition;
  });
};
// 2. Stop recording
const onStopRecord = async () => {
  uri = await audioRecorderPlayer.stopRecorder();
  audioRecorderPlayer.removeRecordBackListener();
};
// 3. Send voice
const file = {
  uri: uri,
  duration: duration,
};
let message = chat.createAudioMessage({
  to: 'user1',
  conversationType: 'C2C',
  payload: {
    file: file,
  },
  // React Native does not support upload progress callback
  // onProgress: function(event) { console.log('file uploading:', event) }
});
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message was successfully sent
  console.log(imResponse);
}).catch(function(imError) {
  // The message failed to be sent
  console.warn('sendMessage error:', imError);
});
```

Note:

To enable `VoiceMessage` to play across all platforms, packaging the iOS app requires specifying VFormatIDKeyIOS as AVEncodingOption.aac for voice recording.

©2013-2025 Tencent Cloud International Pte. Ltd.

To use the album, camera, and microphone, you need to configure the corresponding permissions. After configuring, you need to recompile and run the project to ensure effectiveness.

Android

iOS

In the android directory, find app/src/main/AndroidManifest.xml and add the following permissions:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

In the ios directory, find Info.plist and add the following permissions:

```
<key>NSCameraUsageDescription</key>
<string>We need access to your camera to take photos</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>We need access to your album to select photos</string>
<key>NSMicrophoneUsageDescription</key>
<string>We need access to your microphone to record audio</string>
```

Custom Definition Rich Media Message file Structure

If you choose other third-party plugins to select pictures, videos, files, and record voice, please assemble the file according to the following structure:

Image

Video

File

Voice

uri,fileName,type,width,height is required, fileSize is optional if not available.

```
const file = {
  uri: 'xxx',
  fileName: 'xxx',
  fileSize: 1,
  type: 'xxx',
  width: 1,
  height: 1,
};
```

uri,fileName,type are required, and the type format must be **/*. fileSize and duration are optional if not available.

```
const file = {
  uri: 'xxx',
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  fileName: 'xxx',
  fileSize: 1,
  type: 'xxx',
  duration: 0,
};
```

uri,size,name are required, and size must be greater than 0.

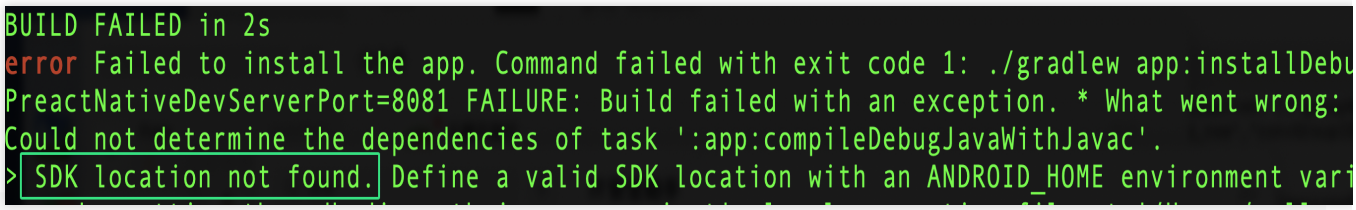
```
const file = {
  uri: 'xxx',
  name: 'xxx',
  size: 1,
};
```

uri,fileName,duration are required. duration is measured in milliseconds (ms). fileSize is optional if not available.

```
const file = {
  uri: 'xxx',
  fileName: 'xxx',
  fileSize: 1,
  duration: 0,
};
```

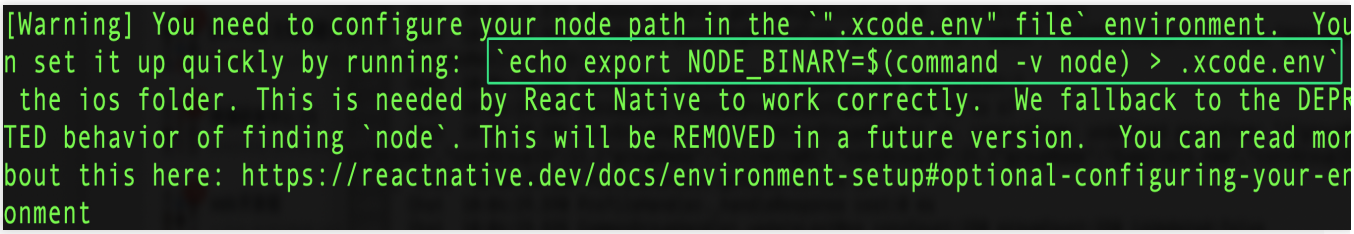
FAQs

1. When running npm run android and an error occurs as shown in the image, please reset the environment variables in the project root directory.




```
export ANDROID_HOME=$HOME/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

2. If executing the Build command in Xcode prompts an issue with the node environment variables, please proceed as follows:




```
cd ios
echo export NODE_BINARY=$(command -v node) > .xcode.env
```

Documentation

For more information on Chat SDK APIs, please refer to the Client API.

Flutter

Last updated：2025-04-16 11:07:25

This document describes how to quickly integrate the Tencent Cloud Chat SDK into your Flutter project.

Environment Requirements

Platform	Version
Flutter	3.0.0 or later
Android	Android Studio 3.5 or later; devices with Android 4.1 or later for apps
iOS	Xcode 11.0 or later. For testing with a real device, ensure that your project has a valid developer signature.

Supported Platforms

We are committed to building a set of Chat SDK and TUIKit for all Flutter platforms, allowing you to run one set of code across all platforms.

Platform	Support or Not
iOS	Supported
Android	Supported
HarmonyOS NEXT	Supported from v8.5.6864+4
macOS	Supported from v4.1.9
Windows	Supported from v4.1.9
Web	Supported from v4.1.1+2
Hybrid development (Adding SDK for Flutter to existing native applications)	Supported from v5.0.0

Note:

For HarmonyOS and web, you need to perform a few extra steps for SDK integration. For details, see Support for the HarmonyOS NEXT and Support for the Flutter for Web in this document.

©2013-2025 Tencent Cloud International Pte. Ltd.

Trying Out Demos

Before integration, you can try out our demos to quickly understand the capabilities of the Tencent Cloud Chat cross-platform SDK and TUIKit for Flutter.

All of the following demos are packaged by the same Flutter project.

Mobile App	Web - HTML5
iOS/Android app	Scan the QR code with your mobile phone to try out

Integrating the Chat SDK

You can directly integrate the Chat SDK for Flutter through pub add, or write the Chat SDK into `pubspec.yaml` .

Installing the Chat SDK via `flutter pub add`

Enter the following command in the terminal window (the Flutter environment is ready):

```
flutter pub add tencent_cloud_chat_sdk
```

Writing the Chat SDK into `pubspec.yaml`

```
dependencies:
# You can check the latest version of the Chat SDK for Flutter on https://pub.dev/p
  tencent_cloud_chat_sdk: "Latest version"
```

Here, your editor may automatically run `flutter pub get` . If not, enter the command `flutter pub get` .

Support for the Flutter for HarmonyOS NEXT

The UI-less SDK (tencent_cloud_chat_sdk) version 8.5.6864+4 and later now supports HarmonyOS NEXT. This implementation is developed based on the HarmonyOS-adapted Flutter 3.22 version.

As HarmonyOS has adapted many Flutter third-party libraries, and the tencent_cloud_chat_sdk only utilizes the path_provider third-party library, you need to add dependency overrides for the HarmonyOS-adapted version of path_provider in your project's root directory pubspec.yaml file.

```
dependency_overrides:
  path_provider:
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    git:
      url: "https://gitee.com/openharmony-sig/flutter_packages.git"
      path: "packages/path_provider/path_provider"
```

Support for the Flutter for Web

Chat SDK (tencent_cloud_chat_sdk) 4.1.1+2 or later provides full compatibility with web.
To enable support for web, you need to perform the following extra steps, compared with the steps to enable support for Android and iOS:

Upgrading to Flutter 3.x

Flutter 3.x has been dramatically optimized for web performance and is highly recommended for Flutter web project development.

Importing JS

Note:
If your existing Flutter project does not support web, run `flutter create .` in the root directory of the project to add web support.
Go to the `web/` directory of your project and use npm or Yarn to install related JS dependencies. To initialize the project, follow the on-screen instructions.

```
cd web

npm init

npm i tim-js-sdk

npm i tim-upload-plugin
```

Open `web/index.html` and import the JS files in `<head> </head>`. See below:

```
<script src="./node_modules/tim-upload-plugin/index.js"></script>
<script src="./node_modules/tim-js-sdk/tim-js-friendship.js"></script>
```

FAQs

What should I do if `flutter pub get/add` fails?

Configure Flutter to use a mirror site as instructed in Flutter.

©2013-2025 Tencent Cloud International Pte. Ltd.

Windows

Last updated：2024-11-04 14:32:15

This document describes how to quickly integrate the Chat SDK into your Windows project.

Environment Requirements

Operating system: Windows 7 or later.
Development environment: Visual Studio 2010 or later. Visual Studio 2019 is recommended.

Integrating Chat SDK

The following describes how to integrate the SDK into a Visual Studio 2019 project by creating a simple MFC project.

Step 1. Download Chat SDK

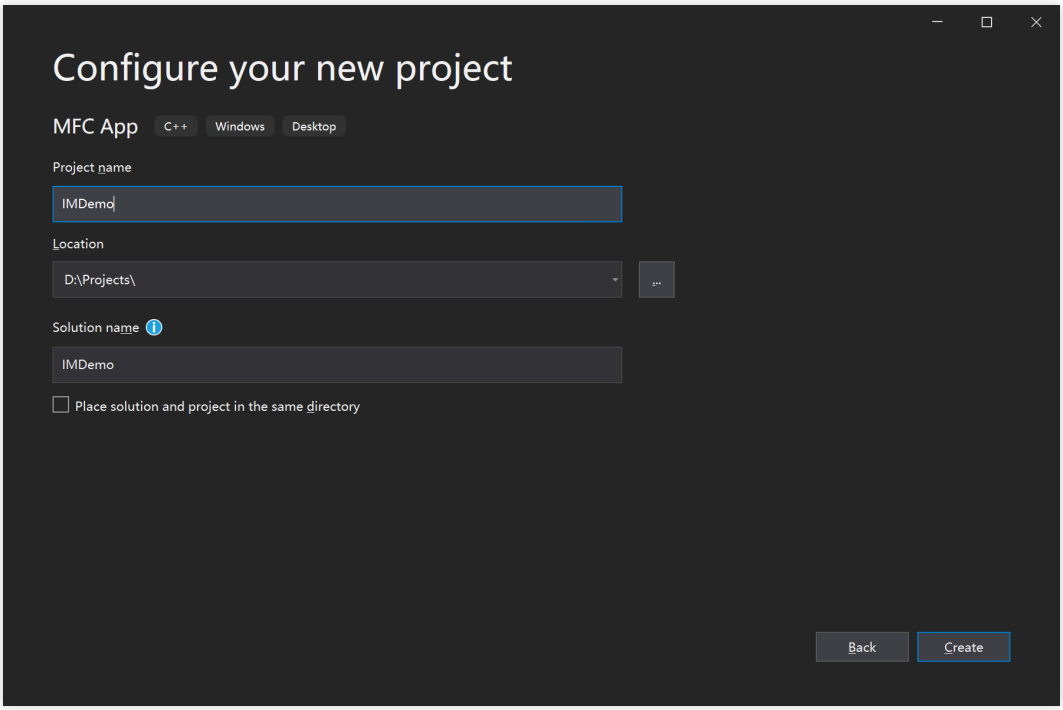
Download the Chat SDK for Windows here, and decompress it. You can rename it "ImSDK" for convenience. It contains the following parts:

Directory Name	Description
c_include	C API header files
cpp_include	C++ API header files
shared_lib\\Win32	**32-bit Release mode**, which uses the /MT option to link library files.
shared_lib\\Win64	**64-bit Release mode**, which uses the /MT option to link library files.

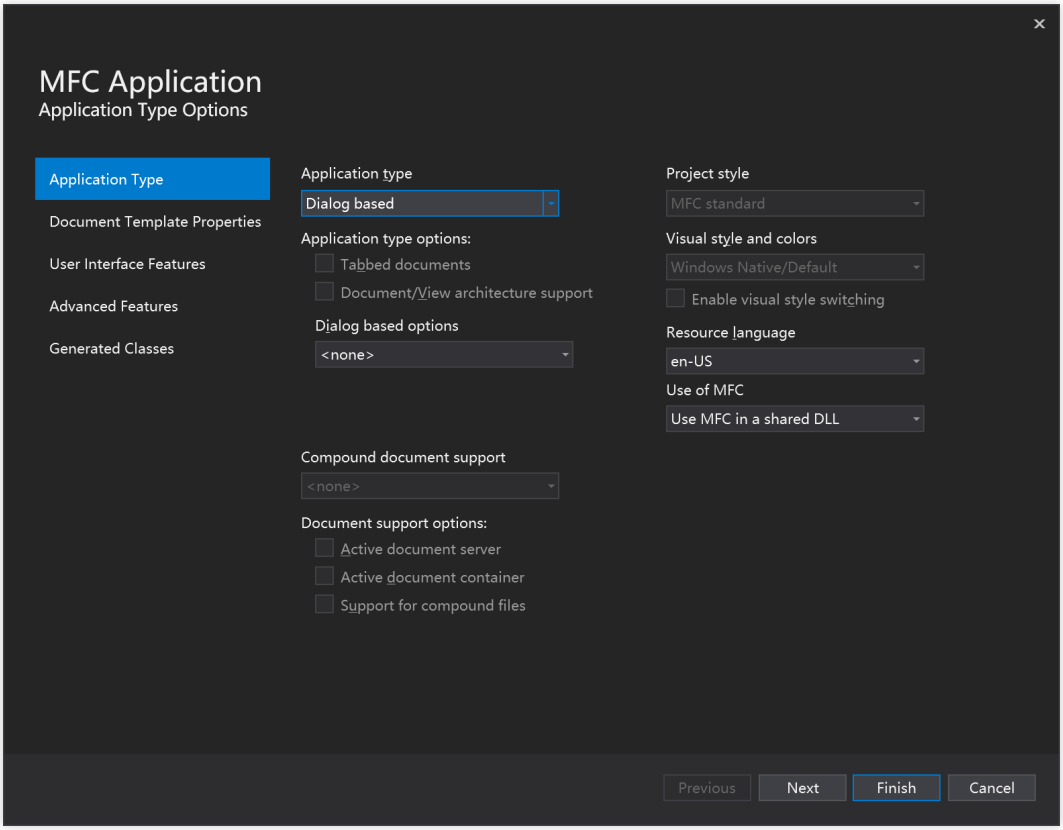
Step 2. Create a project

Open Visual Studio and create an MFC application named "IMDemo". If the MFC application is not one of the first options, you can search for it through the search template at the top.

©2013-2025 Tencent Cloud International Pte. Ltd.



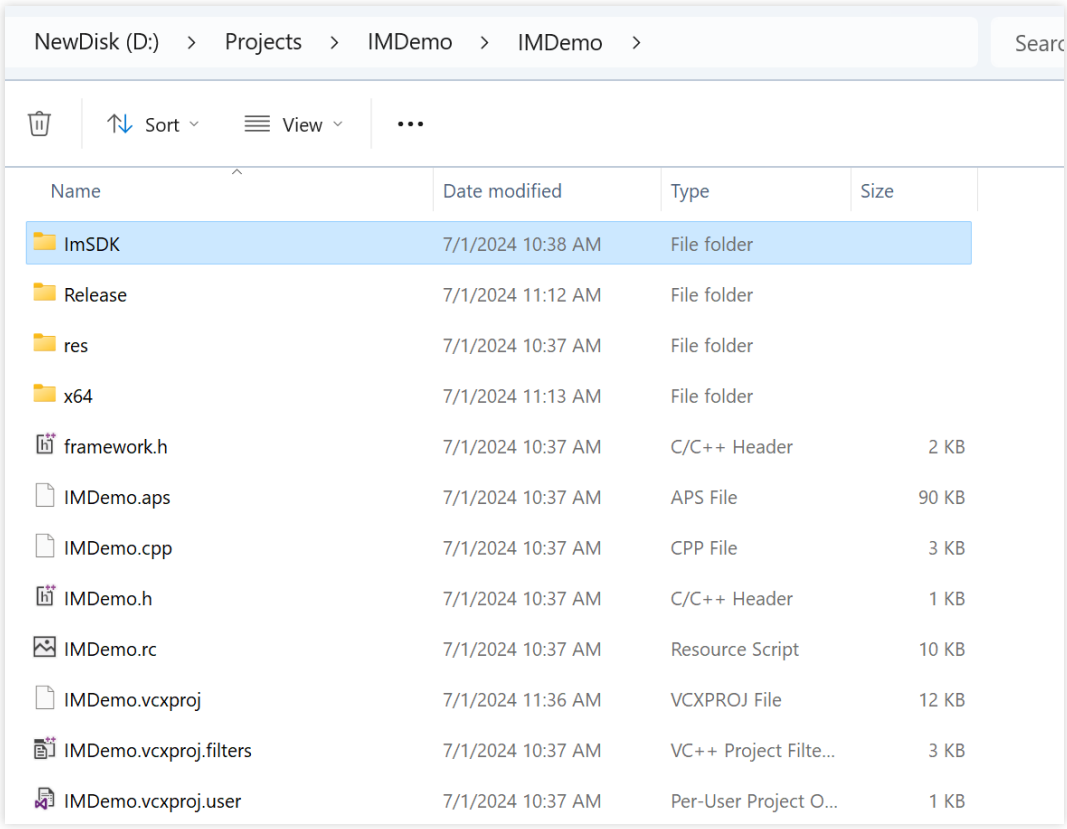
For quick integration, select the relatively simple **Dialog based** type on the **Application Type** page of the wizard. For other configuration items, keep them as default.



©2013-2025 Tencent Cloud International Pte. Ltd.

Step 3. Copy the files

Copy the SDK folder generated after decompression (the "ImSDK" folder obtained in step 1) to the directory where `IMDemo.vcxproj` is located.



Step 4. Modify the project configuration

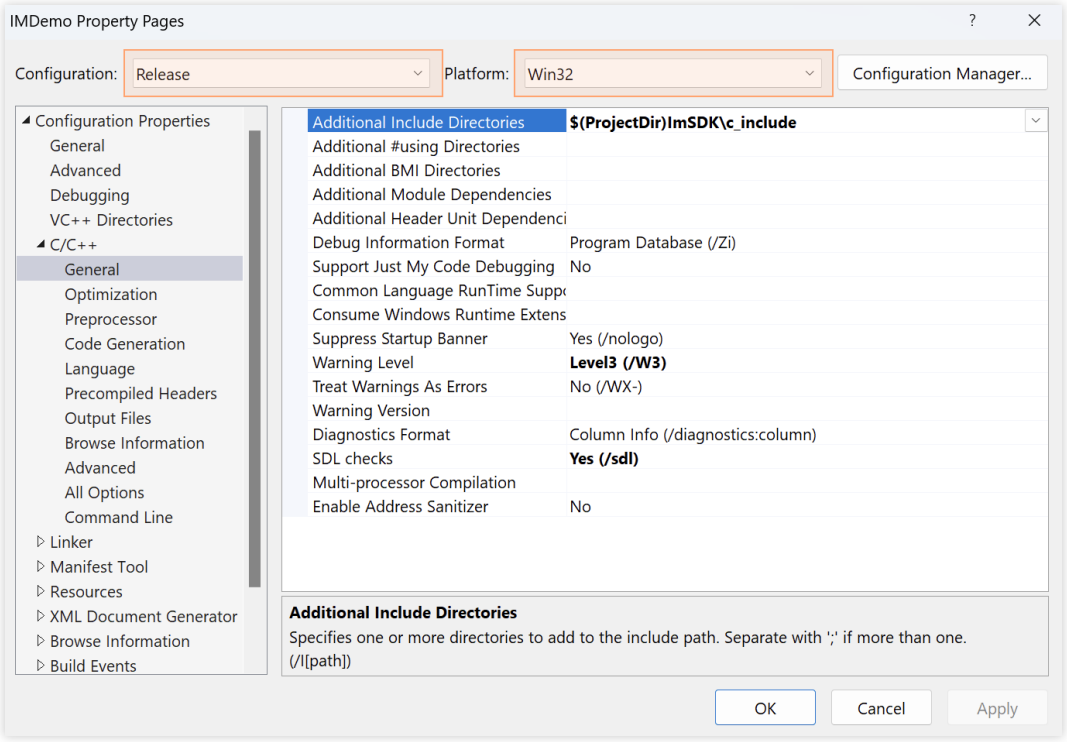
The SDK provides 32-bit and 64-bit static libraries in **Release** mode, which require some special configurations.

Open the IMDemo attribute page, select **Solution Explorer**, right-click `IMDemo`, and select **Properties**.

The following takes the **32-bit Release mode** as an example:

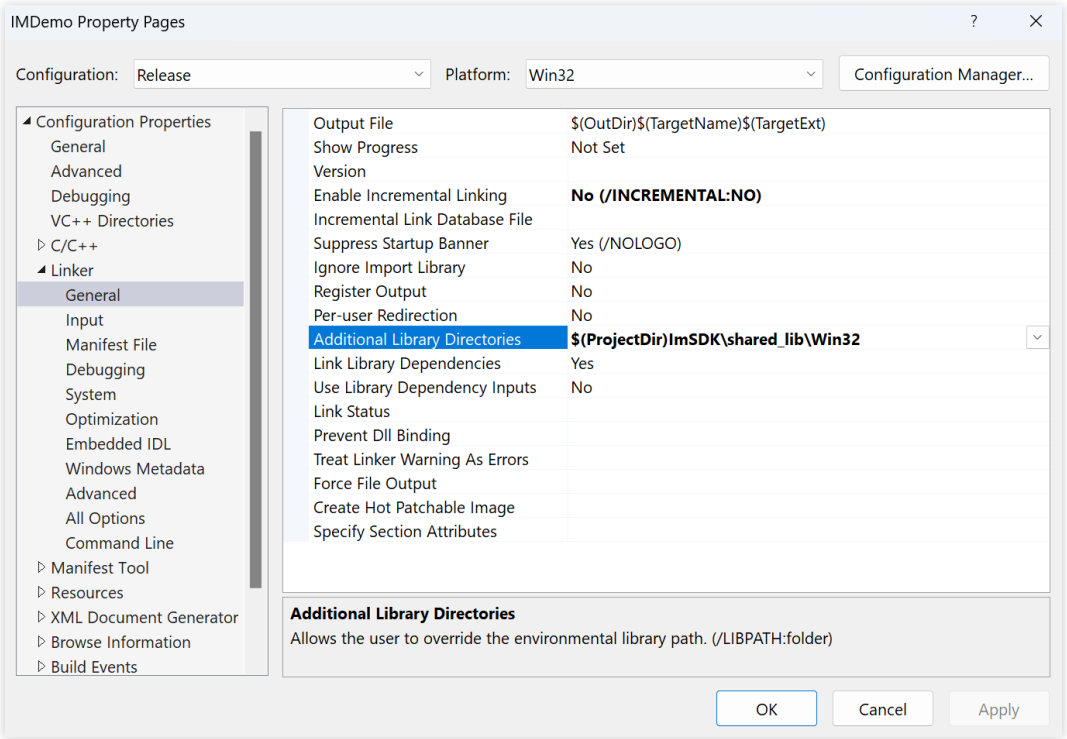
1. Add the include directory.

Go to **C/C++** > **General** > **Additional Include Directories** and add the SDK C header file directory `$(ProjectDir)ImSDK\\c_include`.



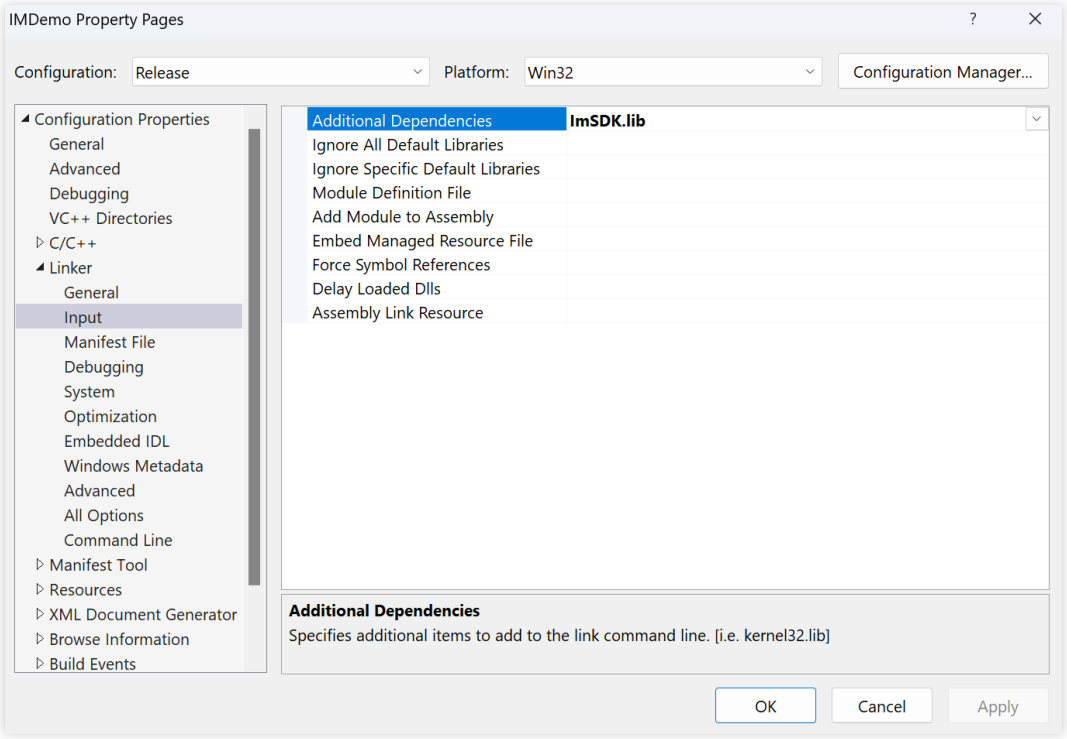
2. Add the library directory.

Go to **Linker** > **General** > **Additional Library Directories** and add the SDK library directory `$(ProjectDir)ImSDK\\shared_lib\\Win32`.



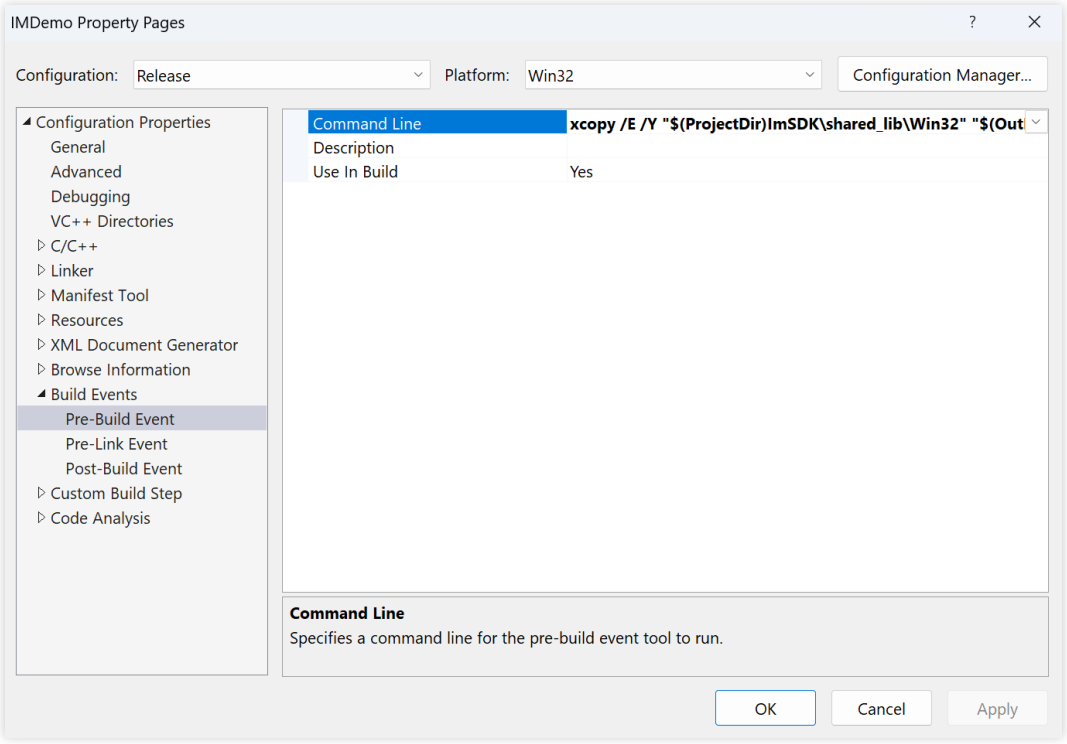
3. Add the library file.

Go to **Linker** > **Input** > **Additional Dependencies** and add the SDK library file `ImSDK.lib` .



4. Copy the DLL file to the execution directory.

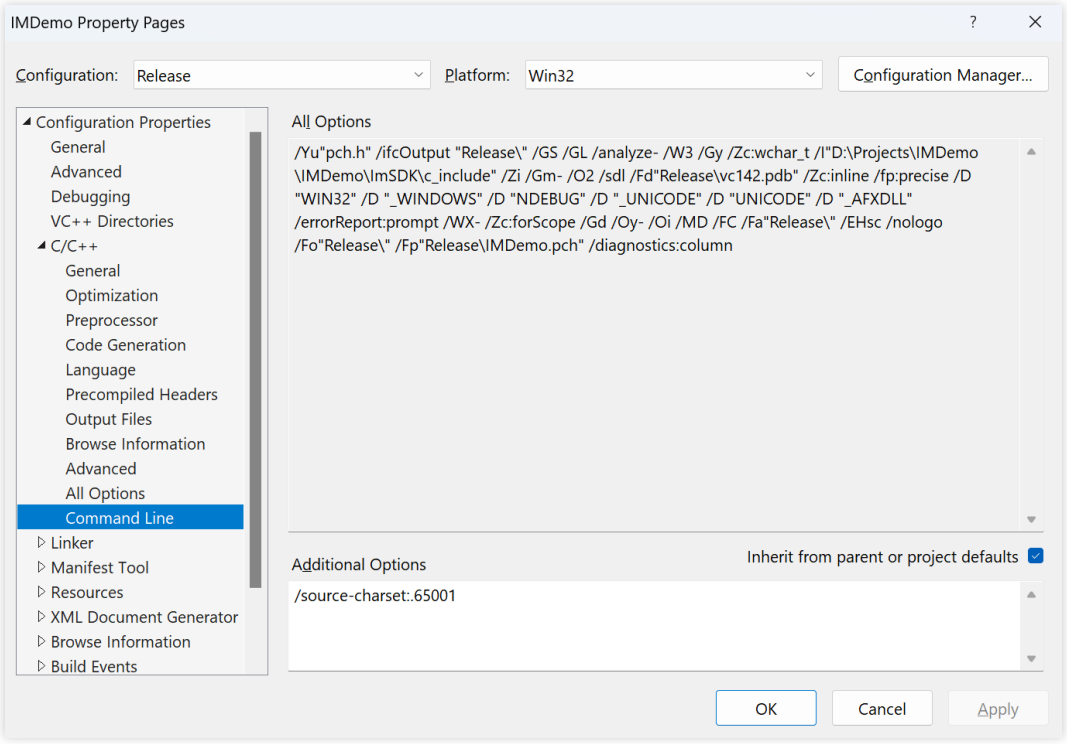
Go to **Build Events** > **Pre-Build Event** > **Command Line**, enter `xcopy /E /Y "$(ProjectDir)ImSDK\\shared_lib\\Win32" "$(OutDir)"` , and copy the `ImSDK.dll` dynamic library file to the output directory of the project.



5. Specify the encoding format of the source file.

The SDK header file uses the UTF-8 encoding format, whereas some compilers compile source files in the default system encoding format. This may lead to compilation failure. Set this parameter to instruct compilers to compile source files using UTF-8 encoding.

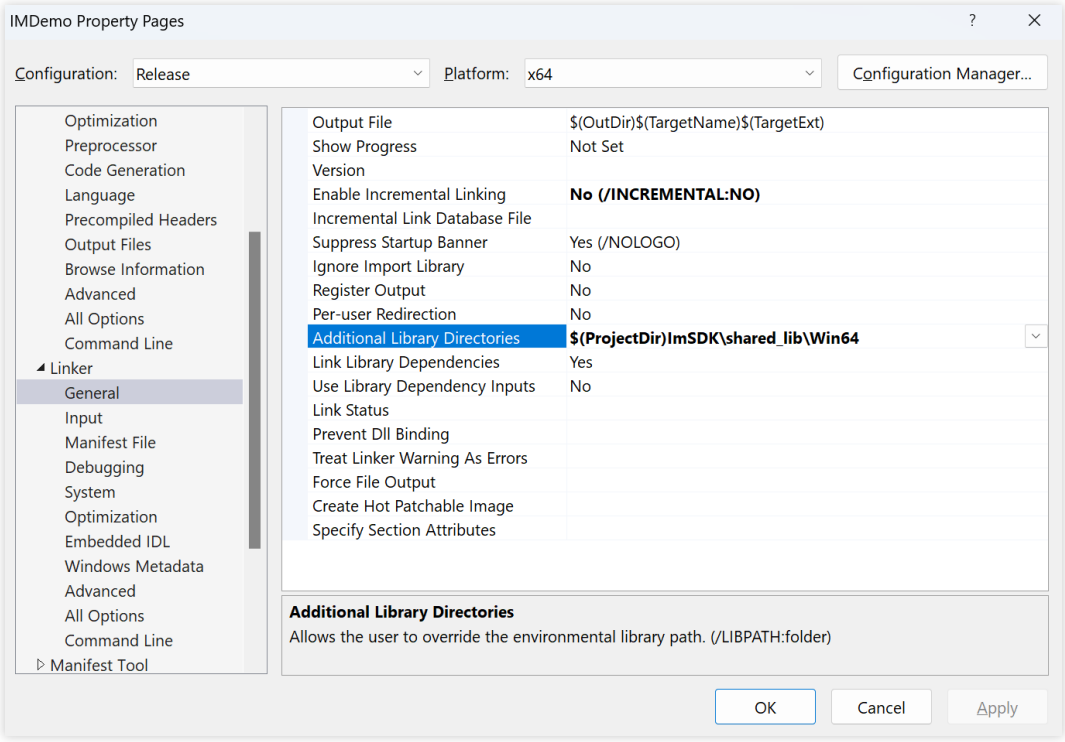
Go to **C/C++** > **Command Line** > **Additional Options** and enter `/source-charset:.65001` .



Most of the settings for the **64-bit Release** are similar to those for the **32-bit Release**. The difference lies in the SDK library directory as follows:

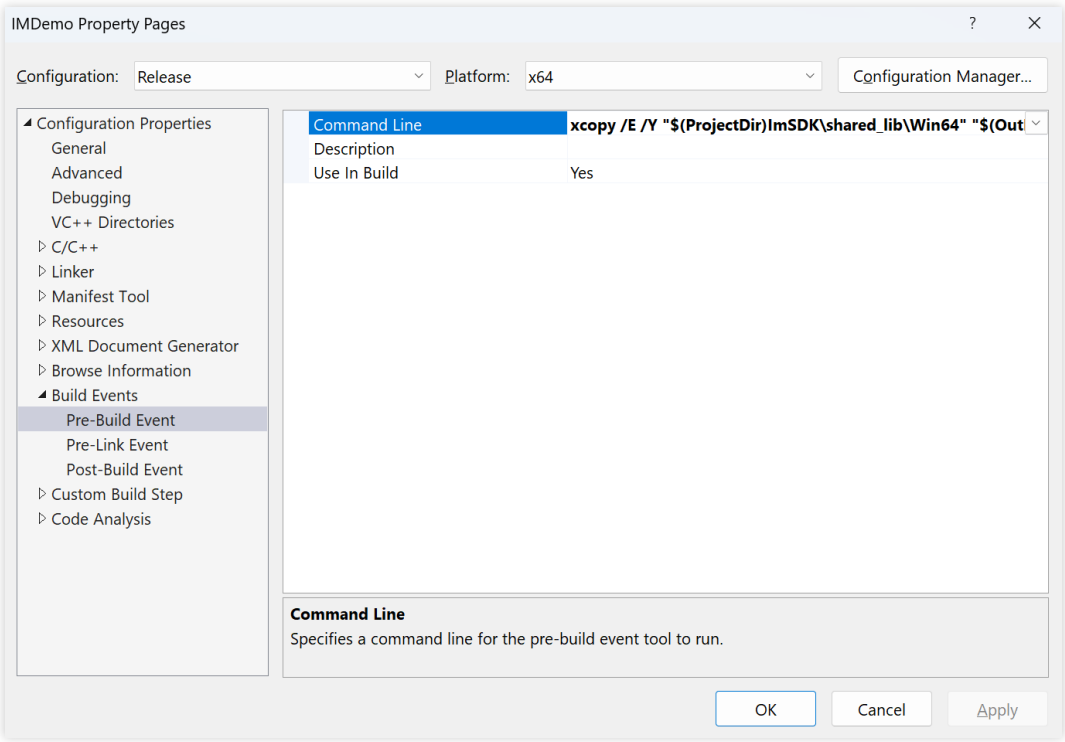
1. Add the library directory.

Go to **Linker** > **General** > **Additional Library Directories** and add the SDK library directory `$(ProjectDir)ImSDK\\shared_lib\\Win64`.



2. Copy the DLL file to the execution directory.

In **Release mode**, go to **Build Events** > **Pre-Build Event** > **Command Line**, enter `xcopy /E /Y "$(ProjectDir)ImSDK\\shared_lib\\Win64" "$(OutDir)"` , and copy the `ImSDK.dll` dynamic library file to the output directory of the project.



Step 5. Print Chat SDK version number

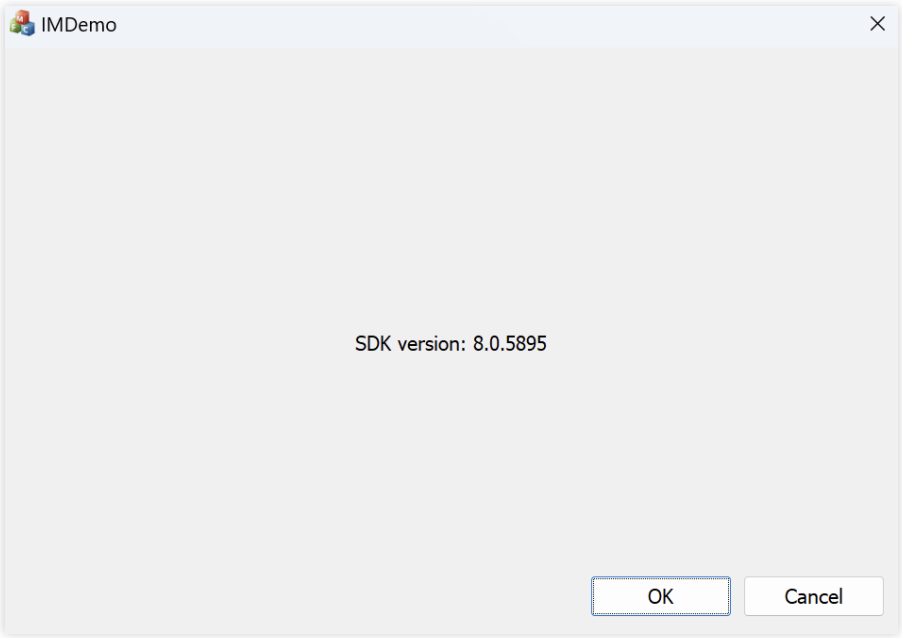
In the `IMDemoDlg.cpp` file, add the header file inclusion:

```
#include "TIMCloud.h"
```

In the `IMDemoDlg.cpp` file, find the `CIMDemoDlg::OnInitDialog` function and add the following test code before `return` :

```
SetWindowText(L"IMDemo");
CString szText;
szText.Format(L"SDK version: %hs", TIMGetSDKVersion());
CWnd* pStatic = GetDlgItem(IDC_STATIC);
pStatic->SetWindowTextW(szText);
```

Press **F5** to run the code and print the SDK version number.



FAQs

If the following error occurs, check whether the directory of the SDK header file has been correctly added according to the preceding project configuration:

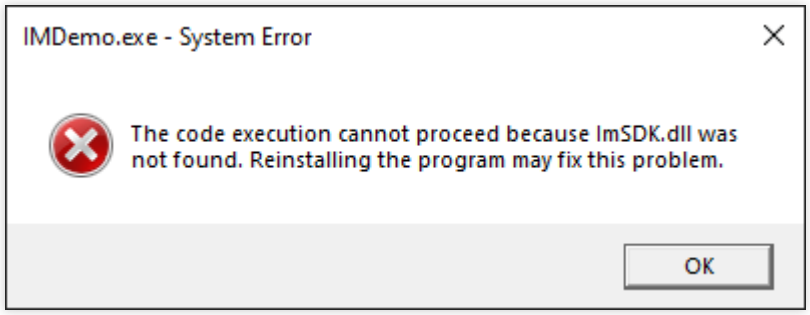
```
fatal error C1083: unable to open the header file: "TIMCloud.h": No such file
or directory
```

If the following error occurs, check whether the SDK library directory and library file have been correctly added according to the preceding project configuration:

```
LINK : fatal error LNK1104: Unable to open the `ImSDK.lib` file.
```

```
error LNK2019: Unresolved external symbol `__imp__TIMGetSDKVersion`, referenced in the `"protected: virtual int __thiscall CIMDemoDlg::OnInitDialog(void)" (?OnInitDialog@CIMDemoDlg@@MAEHXZ)` function
```

If an error occurs, check whether the SDK's DLL has been copied to the execution directory as described in the project configuration step above.



©2013-2025 Tencent Cloud International Pte. Ltd.

Mac

Last updated：2024-11-04 14:34:26

This document describes how to quickly integrate the Chat SDK (Mac) into your projects. To configure and integrate the SDK, follow these steps.

Development Environment Requirements

Xcode 9.0+.
Mac device running OS X 10.10 or later.
The project has been configured with a valid developer signature.

Integrating Chat SDK

You can either automatically integrate the Chat SDK by using CocoaPods, or manually download the SDK and import it to your current project.

Automatically loading CocoaPods

1. Install CocoaPods

Run the following command in a terminal window (you need to install the Ruby environment on your Mac device in advance):

```
sudo gem install cocoapods
```

2. Create a Podfile

Navigate to the path where the project is located and run the following command. Then, a Podfile will appear under the project path.

```
pod init
```

3. Edit the Podfile

Edit the Podfile as follows:

```
platform :macos, '10.10'
source 'https://github.com/CocoaPods/Specs.git'
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
target 'mac_test' do
pod 'TXIMSDK_Mac'
end
```

4. Update and install the SDK

Run the following command in a terminal window to update the local library file and install TXIMSDK_Mac:

```
pod install
```

Alternatively, run the following command to update the local library version:

```
pod update
```

After the pod command is executed, a project file integrated with the SDK and suffixed .xcworkspace will be generated. Double-click the file to open it.

Manual integration

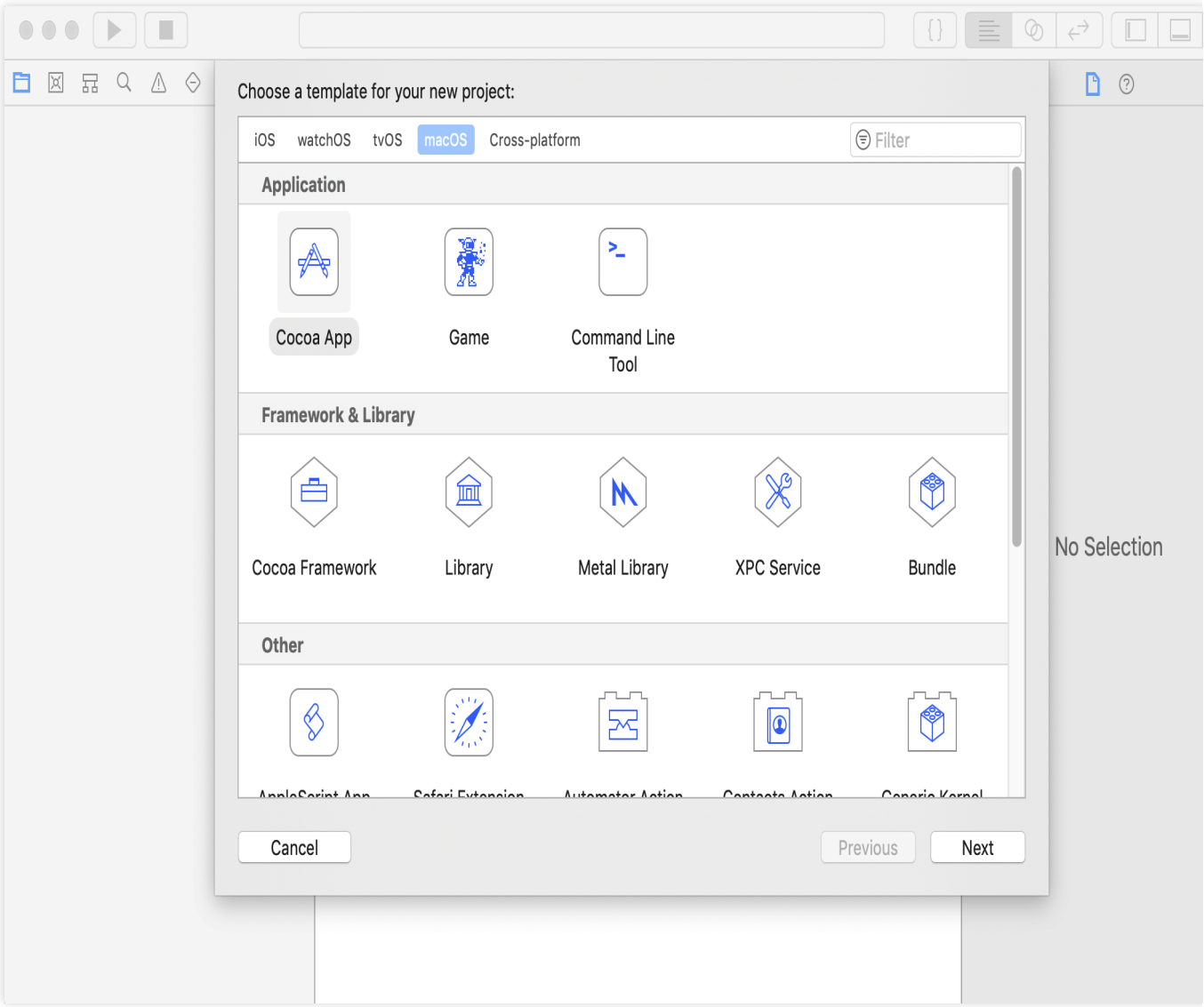
1. Obtain the SDK download URL from Github:

ImSDKForMac.framework is the core dynamic library file of the SDK.

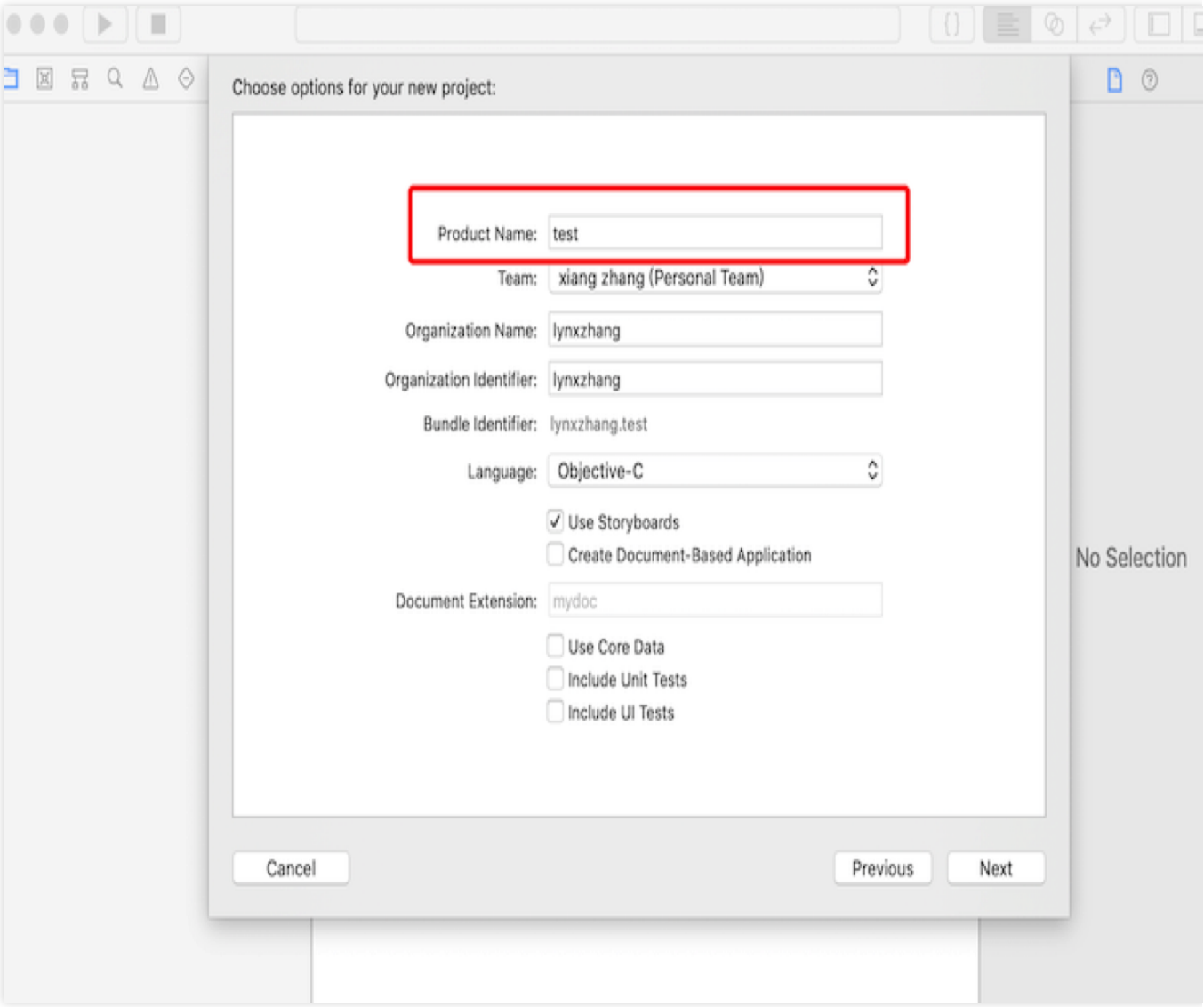
Pack Name	Description	ipa Increment
ImSDKForMac.framework	Chat feature pack	1.4 MB

2. Create a project

Create a project:



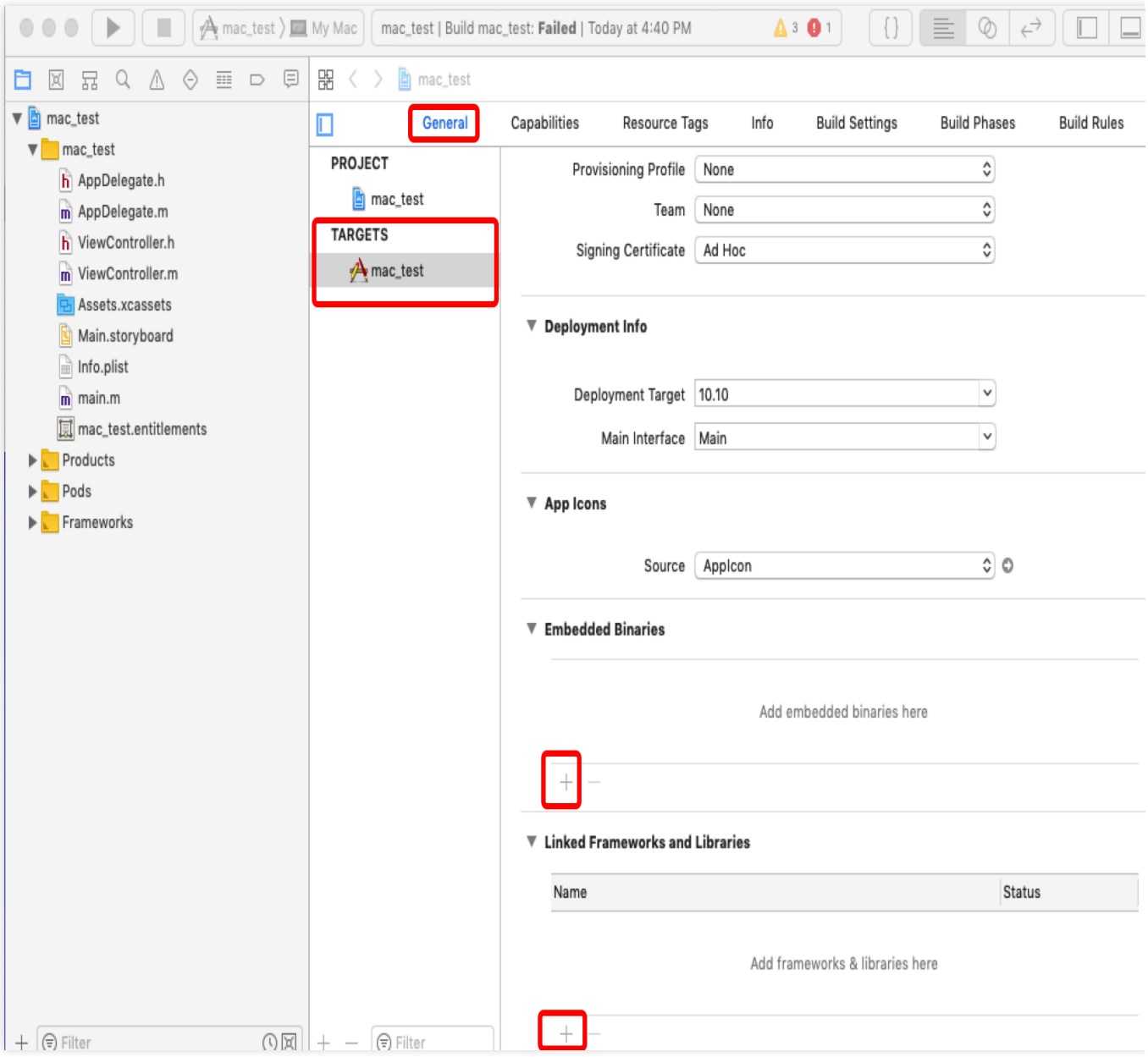
Enter a project name:



2. Integrate Chat SDK

Add the dependent library: select the **Target** of Demo. On the **General** panel, add the dependent library under **Embedded Binaries** and **Linked Frameworks and Libraries**.

©2013-2025 Tencent Cloud International Pte. Ltd.



Add the dependent library:

```
ImSDKForMac.framework
```

Caution:

You need to add `-ObjC` in **Build Setting** > **Other Linker Flags**.

Referencing Chat SDK

Use the SDK in project code in two ways:

Method 1: Navigate to **Xcode** > **Build Setting** > **Header Search Paths**, and set the ImSDKForMac.framework/Headers path. In files that require the SDK API, directly reference the header file

"ImSDK.h".

```
#import "ImSDK.h"
```

Method 2: in the files that require the SDK API, import the header file <ImSDKForMac/ImSDK.h>.

```
#import <ImSDKForMac/ImSDK.h>
```

Unity

Last updated：2024-11-04 14:34:26

This document describes how to quickly integrate the Chat SDK for Unity to your projects. To configure and integrate the SDK, follow these steps.

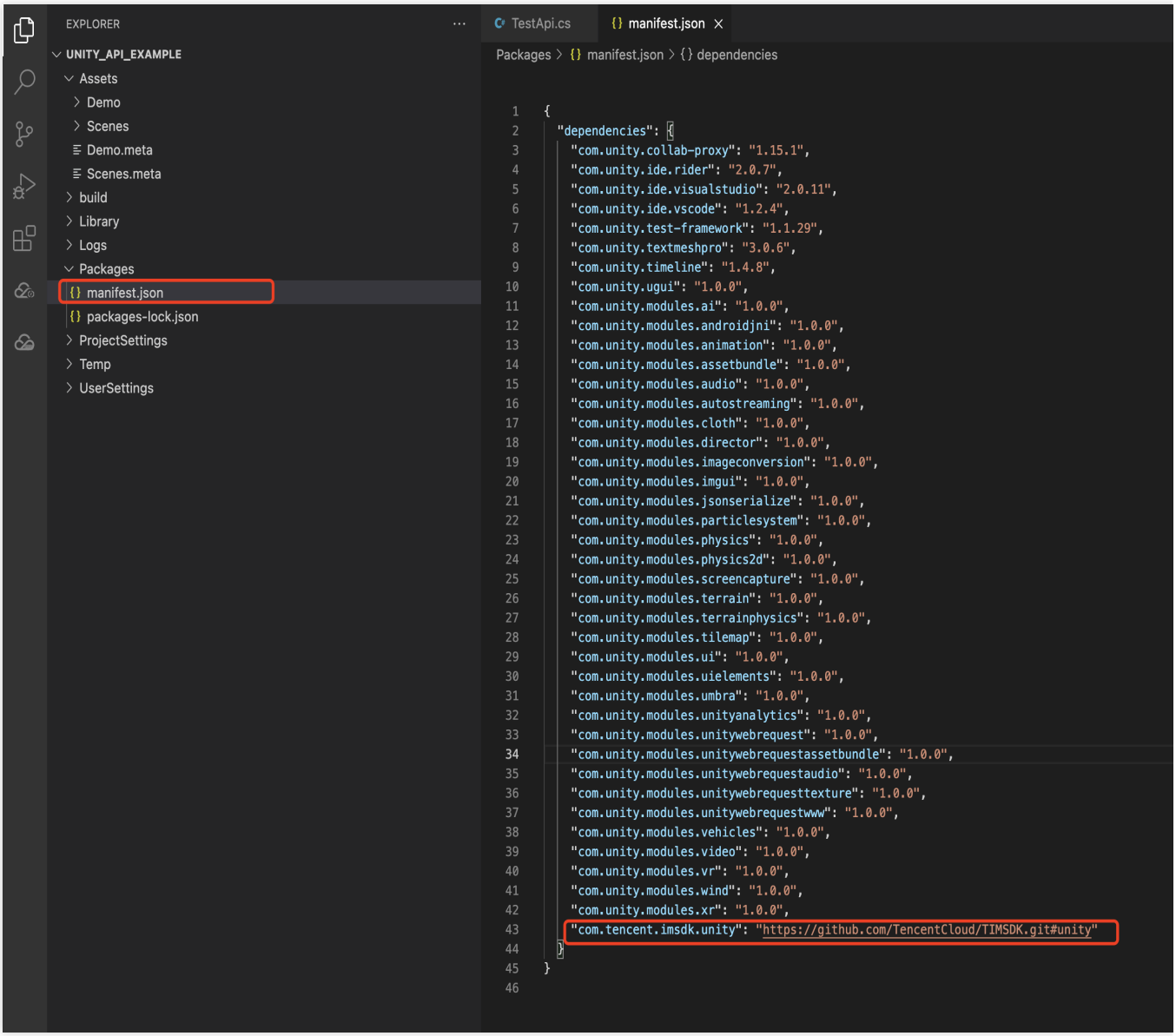
Environment Requirements

Platform	Version
Unity	2019.4.15f1 or later
Android	Android Studio 3.5 or later; devices with Android 4.1 or later for apps
iOS	Xcode 11.0 or later. Ensure that your project has a valid developer signature.

UPM Integration (Recommended)

1. Find the `manifest.json` file:

©2013-2025 Tencent Cloud International Pte. Ltd.

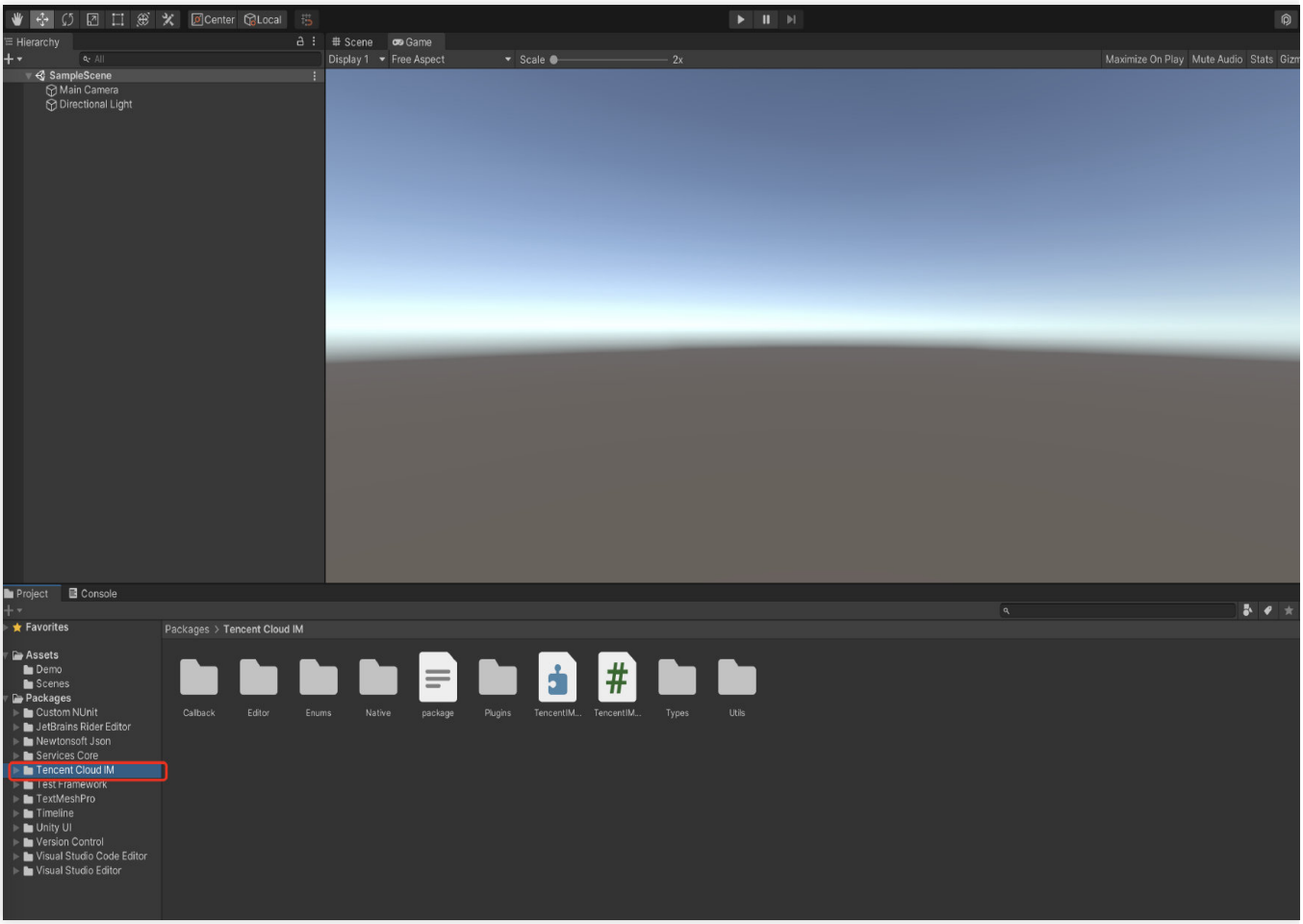


2. Modify it as follows:

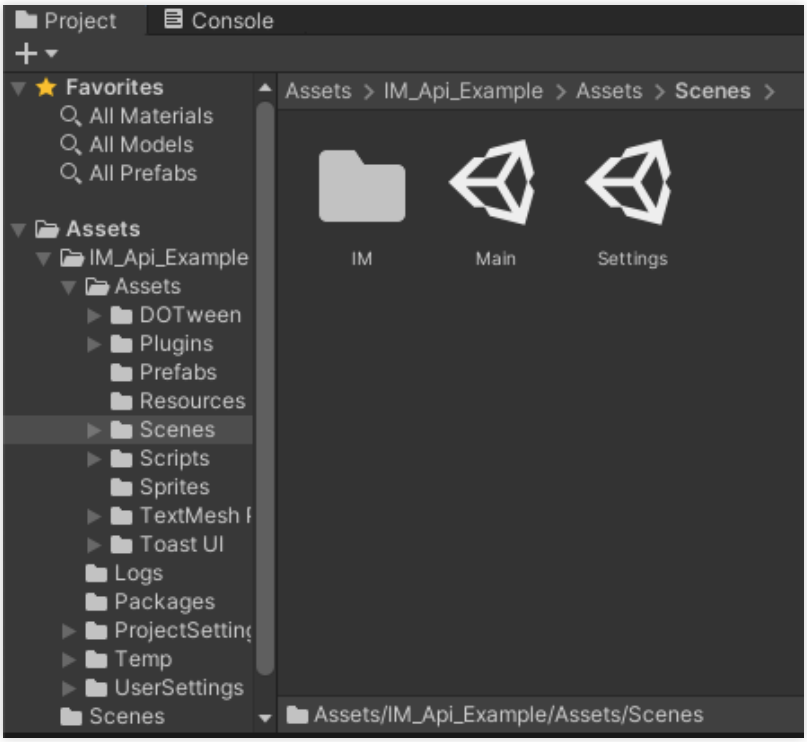
```
{
  "dependencies":{
    "com.tencent.imsdk.unity":"https://github.com/TencentCloud/chat-sdk-unity.git#u
  }
}
```

3. Open the project in the Unity Editor, wait until dependencies are loaded, and confirm the Tencent Cloud Chat is successfully loaded.

©2013-2025 Tencent Cloud International Pte. Ltd.



4. This is a test step. You can download IM_Api_Example and put it in your project after decompression.

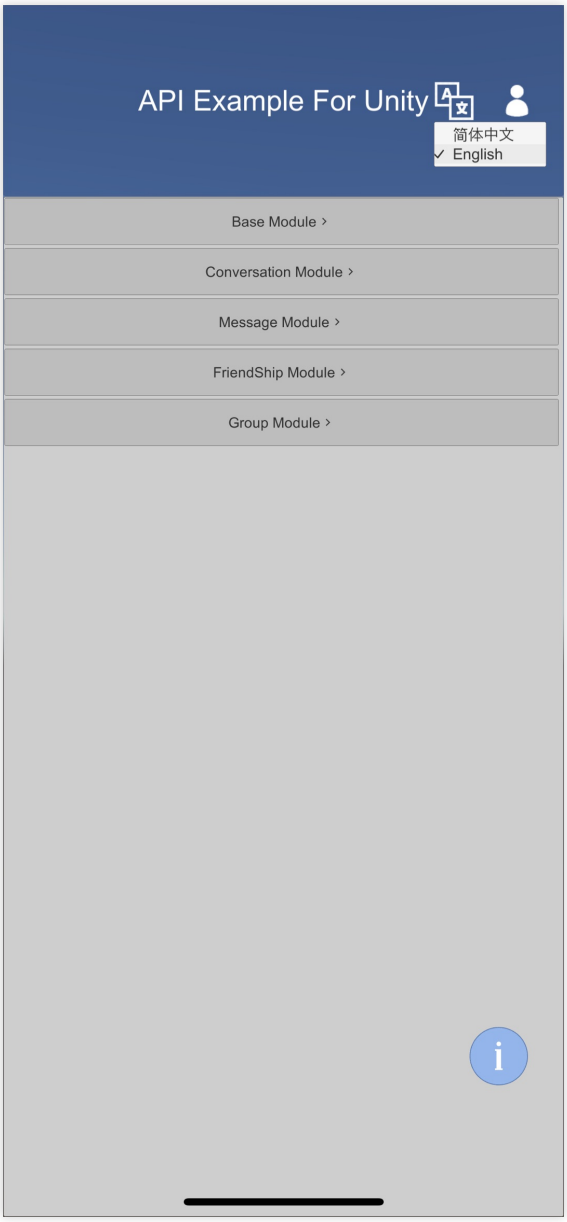


Note:

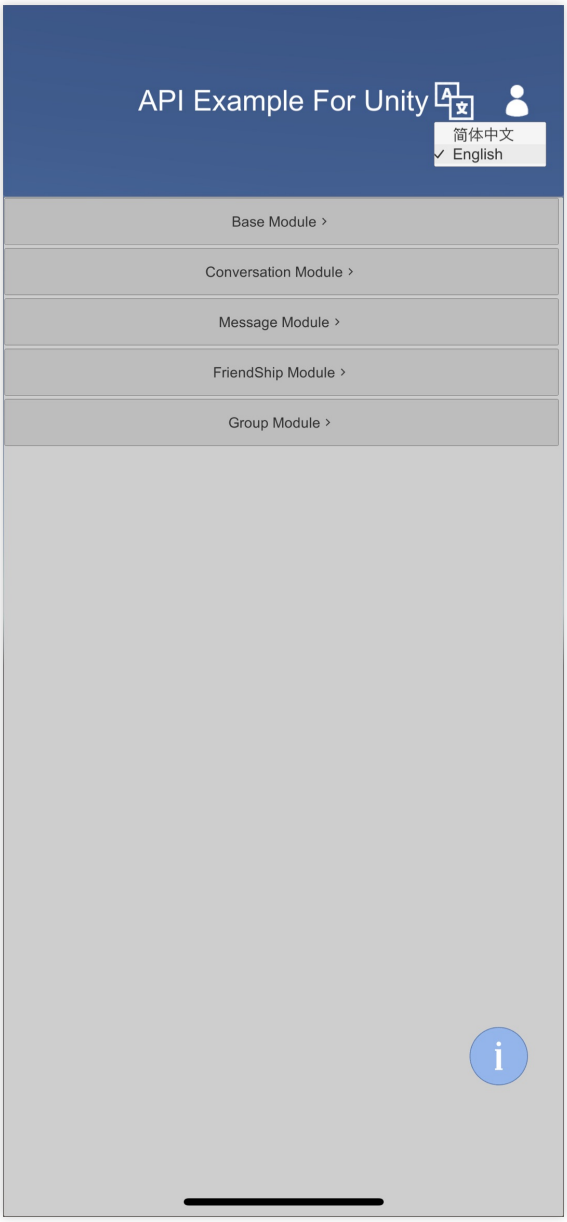
IM_Api_Example is a demo used to test the callback data of the SDK's API. You can also call the API during early project development to manipulate your project.

Drag all the scenes under the `IM_Api_Example/Assets` folder to `Build Settings`, and make sure that the `Main` scene is put at the first place.

©2013-2025 Tencent Cloud International Pte. Ltd.



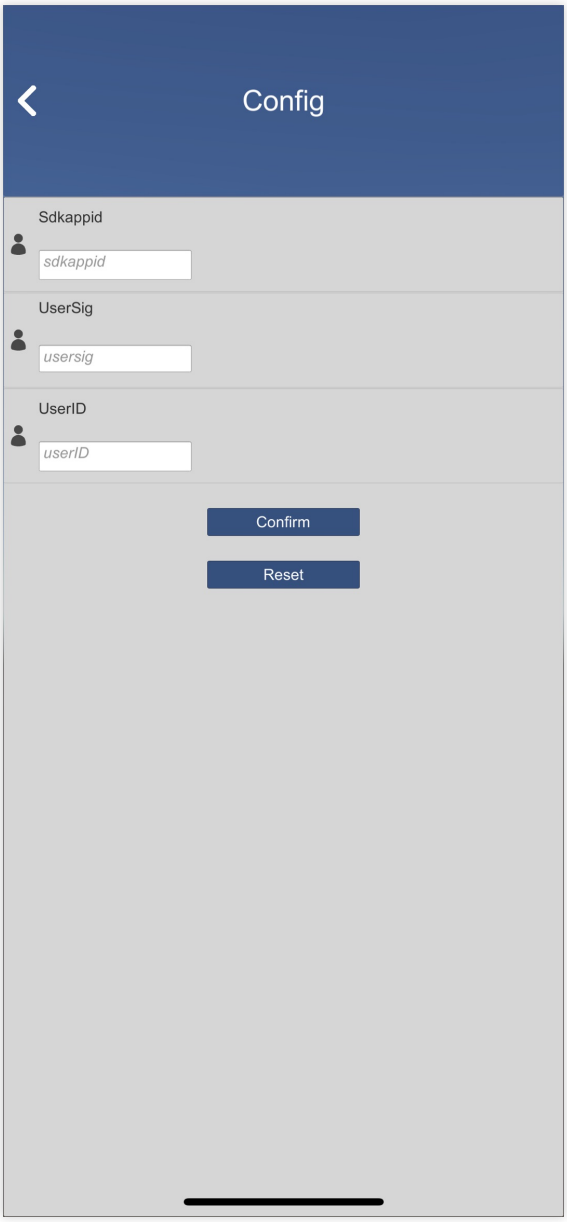
Double-click the `Main` scene under `IM_Api_Example/Assets` to start the demo. Here, you can select the language.



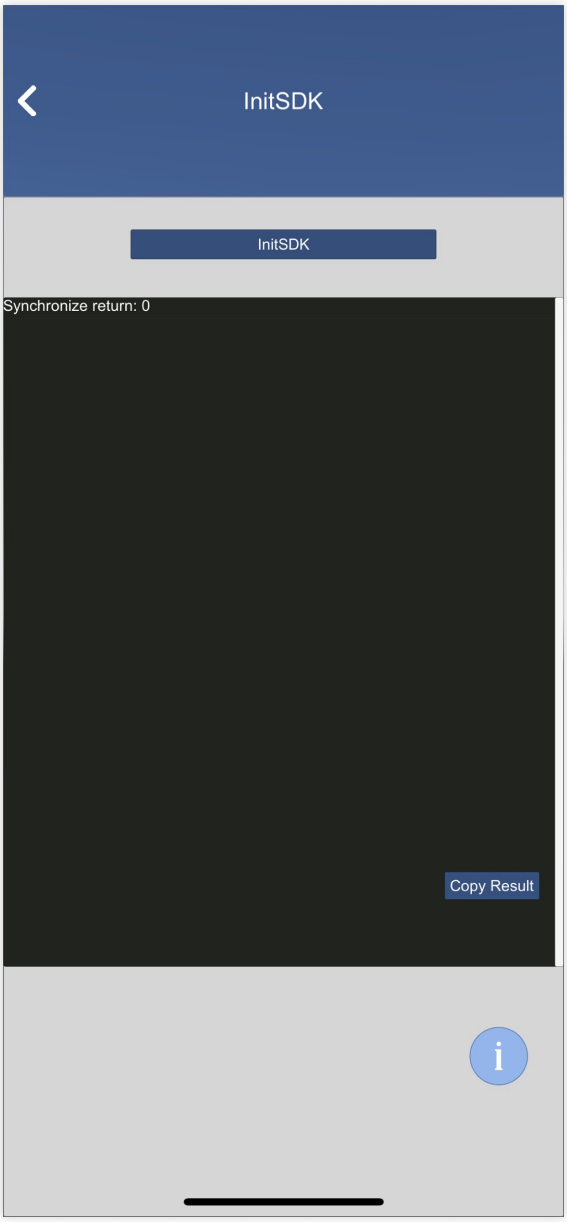
Click

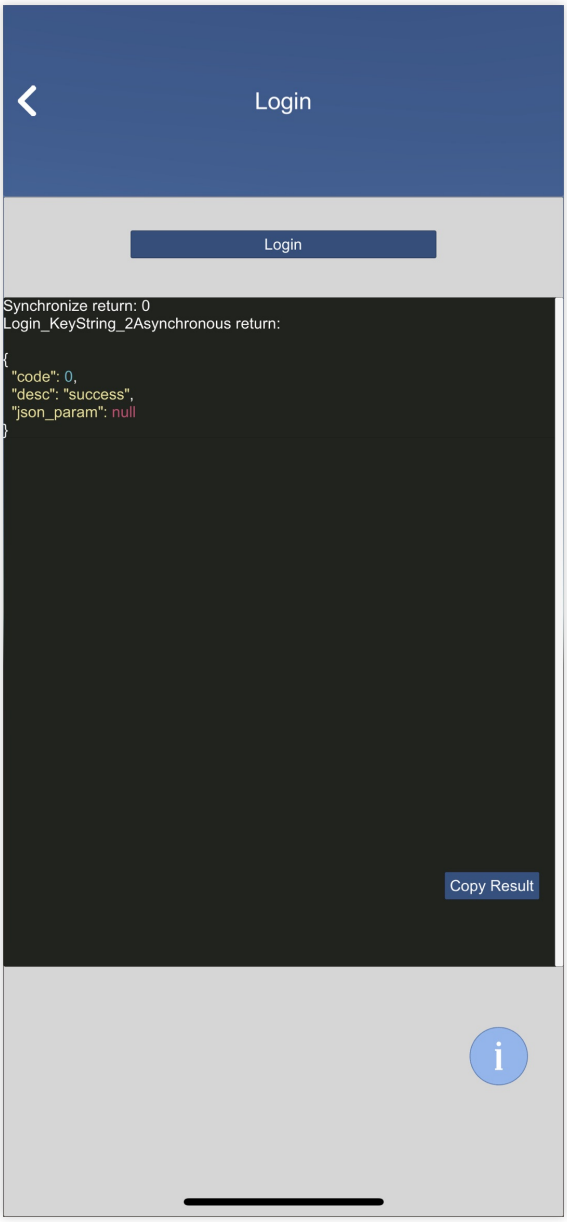


on the right of the header and enter the information.



Click **InitSDK** and **Login** in **Base Module** for initialization and login. Then you can call the APIs available in IM_Api_Example.


InitSDK
InitSDK
Synchronize return: 0
Copy Result


Login
Login
Synchronize return: 0
Login_KeyString_2Asynchronous return:
{
"code": 0,
"desc": "success",
"json_param": null
}
Copy Result

Unreal Engine

Last updated：2024-11-04 14:34:26

This document describes how to quickly integrate the Chat SDK (Unreal Engine) to your projects. To configure and integrate the SDK, follow these steps.

Environment Requirements

Unreal Engine 4.27.1 or later.

Platform	Environment
Android	Android Studio 4.0 or later. Visual Studio 2017 15.6 or later. A real device for testing.
iOS & macOS	Xcode 11.0 or later. OSX 10.11 or later. Ensure your project has a valid developer signature.
Windows	OS: Windows 7 SP1 or later (64-bit based on x86-64). Disk capacity: at least 1.64 GB of space after the IDE and required tools are installed. Install Visual Studio 2019.

Integrating the SDK

1. Download the SDK and SDK source code.
2. Copy the `SDK` folder to the **Source**/**[project_name]** directory of your project (**[project_name]** is the name of your project).
3. Add the following function to the **[project_name].Build.cs** file in your project.

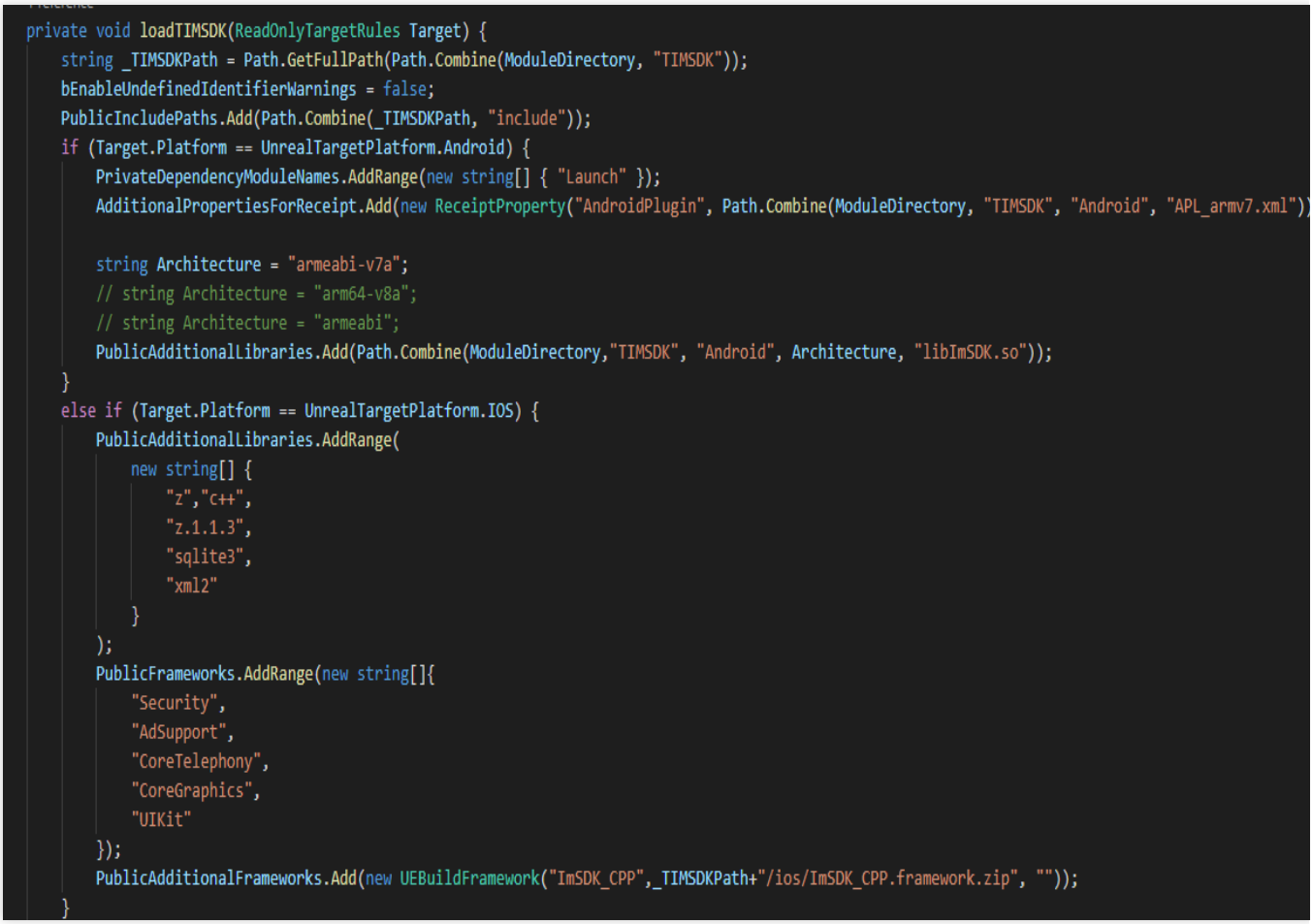
```
// Load the Chat underlying libraries of platforms
private void loadTIMSDK(ReadOnlyTargetRules Target) {
 string _TIMSDKPath = Path.GetFullPath(Path.Combine(ModuleDirectory,
"TIMSDK"));
 bEnableUndefinedIdentifierWarnings = false;
 PublicIncludePaths.Add(Path.Combine(_TIMSDKPath, "include"));
 if (Target.Platform == UnrealTargetPlatform.Android) {
```

```
        PrivateDependencyModuleNames.AddRange(new string[] { "Launch" });
        AdditionalPropertiesForReceipt.Add(new ReceiptProperty("AndroidPlugin",
Path.Combine(ModuleDirectory, "TIMSDK", "Android", "APL_armv7.xml")));

        string Architecture = "armeabi-v7a";
        // string Architecture = "arm64-v8a";
        // string Architecture = "armeabi";
        PublicAdditionalLibraries.Add(Path.Combine(ModuleDirectory,"TIMSDK",
"Android", Architecture, "libImSDK.so"));
    }
    else if (Target.Platform == UnrealTargetPlatform.IOS) {
        PublicAdditionalLibraries.AddRange(
            new string[] {
                "z","c++",
                "z.1.1.3",
                "sqlite3",
                "xml2"
            }
        );
    PublicFrameworks.AddRange(new string[]{
            "Security",
            "AdSupport",
            "CoreTelephony",
            "CoreGraphics",
            "UIKit"
        });
        PublicAdditionalFrameworks.Add(new
UEBuildFramework("ImSDK_CPP",_TIMSDKPath+"/ios/ImSDK_CPP.framework.zip", ""));
    }
    else if(Target.Platform == UnrealTargetPlatform.Mac) {
        PublicAdditionalLibraries.AddRange(new string[] {
            "resolv",
            "z",
            "c++",
            "bz2",
            "sqlite3",
        });
    PublicFrameworks.AddRange(
            new string[] {
                "AppKit",
                "Security",
                "CFNetwork",
                "SystemConfiguration",
            });
        PublicFrameworks.Add(Path.Combine(_TIMSDKPath, "Mac",
"Release","ImSDKForMac_CPP.framework"));
    }
```

```
 else if (Target.Platform == UnrealTargetPlatform.Win64) {
     PublicAdditionalLibraries.Add(Path.Combine(_TIMSDKPath, "win64",
"Release","ImSDK.lib"));
     PublicDelayLoadDLLs.Add(Path.Combine(_TIMSDKPath, "win64", "Release",
"ImSDK.dll"));
     RuntimeDependencies.Add("$(BinaryOutputDir)/ImSDK.dll",
Path.Combine(_TIMSDKPath, "win64", "Release", "ImSDK.dll"));
 }
}
```

4. Call the function in the **[project_name].Build.cs** file:



```
private void loadTIMSDK(ReadOnlyTargetRules Target) {
    string _TIMSDKPath = Path.GetFullPath(Path.Combine(ModuleDirectory, "TIMSDK"));
    bEnableUndefinedIdentifierWarnings = false;
    PublicIncludePaths.Add(Path.Combine(_TIMSDKPath, "include"));
    if (Target.Platform == UnrealTargetPlatform.Android) {
        PrivateDependencyModuleNames.AddRange(new string[] { "Launch" });
        AdditionalPropertiesForReceipt.Add(new ReceiptProperty("AndroidPlugin", Path.Combine(ModuleDirectory, "TIMSDK", "Android", "APL_armv7.xml"))

        string Architecture = "armeabi-v7a";
        // string Architecture = "arm64-v8a";
        // string Architecture = "armeabi";
        PublicAdditionalLibraries.Add(Path.Combine(ModuleDirectory,"TIMSDK", "Android", Architecture, "libImSDK.so"));
    }
    else if (Target.Platform == UnrealTargetPlatform.IOS) {
        PublicAdditionalLibraries.AddRange(
            new string[] {
                "z","C++",
                "z.1.1.3",
                "sqlite3",
                "xml2"
            }
        );
        PublicFrameworks.AddRange(new string[]{
            "Security",
            "AdSupport",
            "CoreTelephony",
            "CoreGraphics",
            "UIKit"
        });
        PublicAdditionalFrameworks.Add(new UEBuildFramework("ImSDK_CPP",_TIMSDKPath+"/ios/ImSDK_CPP.framework.zip", ""));
    }
```

5. You have successfully integrated the Chat SDK. You can use Chat features in your CPP file via `#include "V2TIMManager.h"`.

```
// Get the SDK singleton object
V2TIMManager* timInstance = V2TIMManager::GetInstance();
// Get the SDK version number
V2TIMString timString = timInstance->GetVersion();
// Initialize the SDK
V2TIMSDKConfig timConfig;
timConfig.initPath = static_cast<V2TIMString>("D:\\\\");
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
timConfig.logPath = static_cast<V2TIMString>("D:\\\\");
bool isInit = timInstance->InitSDK(SDKAppID, timConfig);
```

Packaging

macOS

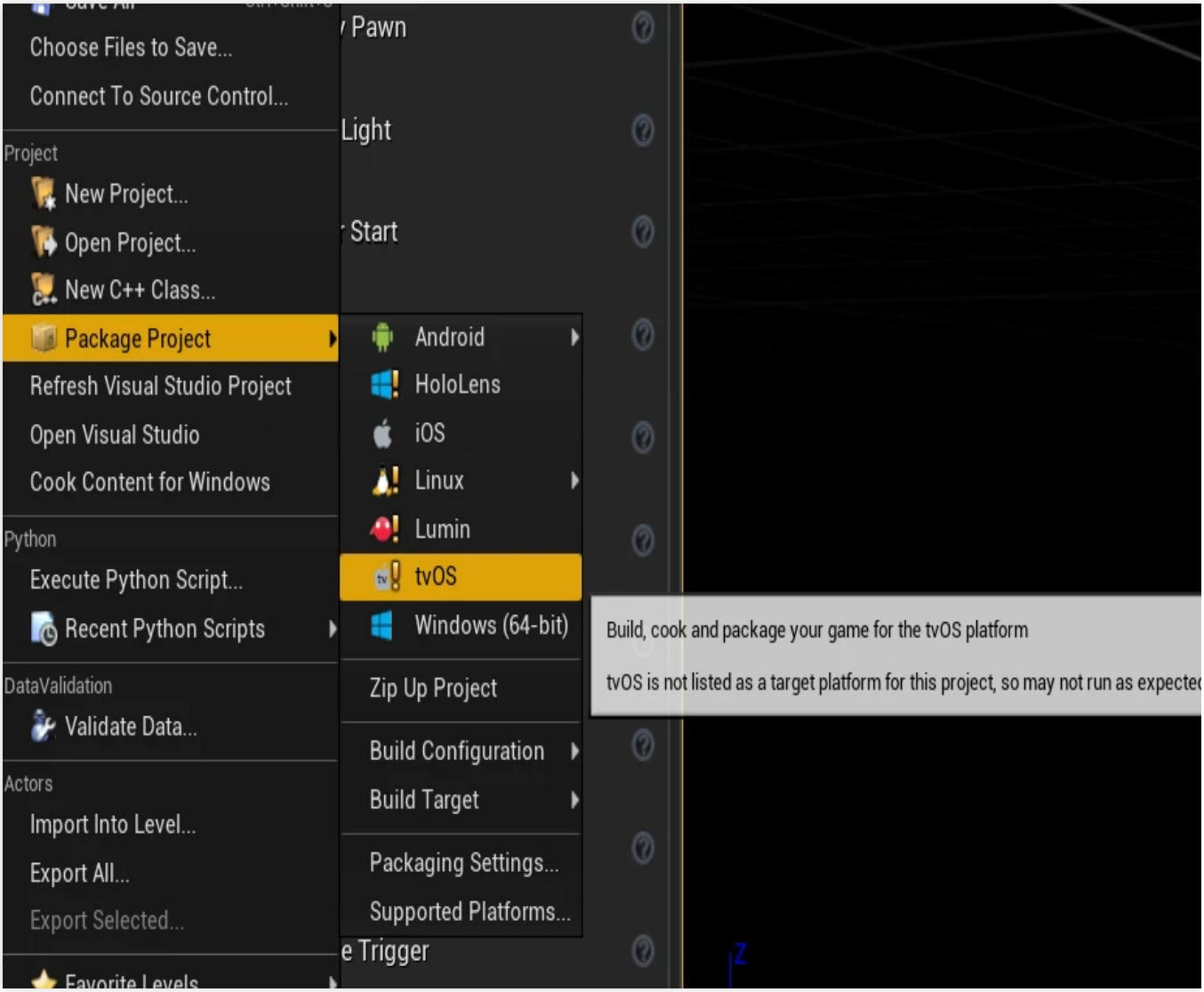
Windows

iOS

Android

File -> **Package Project** -> **Mac**

File -> **Package Project** -> **Windows** -> **Windows(64-bit)**



Go to **File > Package Project > iOS** to package your project

For development and testing, see Android Quick Start.

For packaging, see Packaging Android Projects.

API Documentation of Chat Unreal Engine

For more information on APIs, see API Overview.

Initialization Android&iOS&Windows&Mac

Last updated：2025-01-06 17:39:50

Feature Description

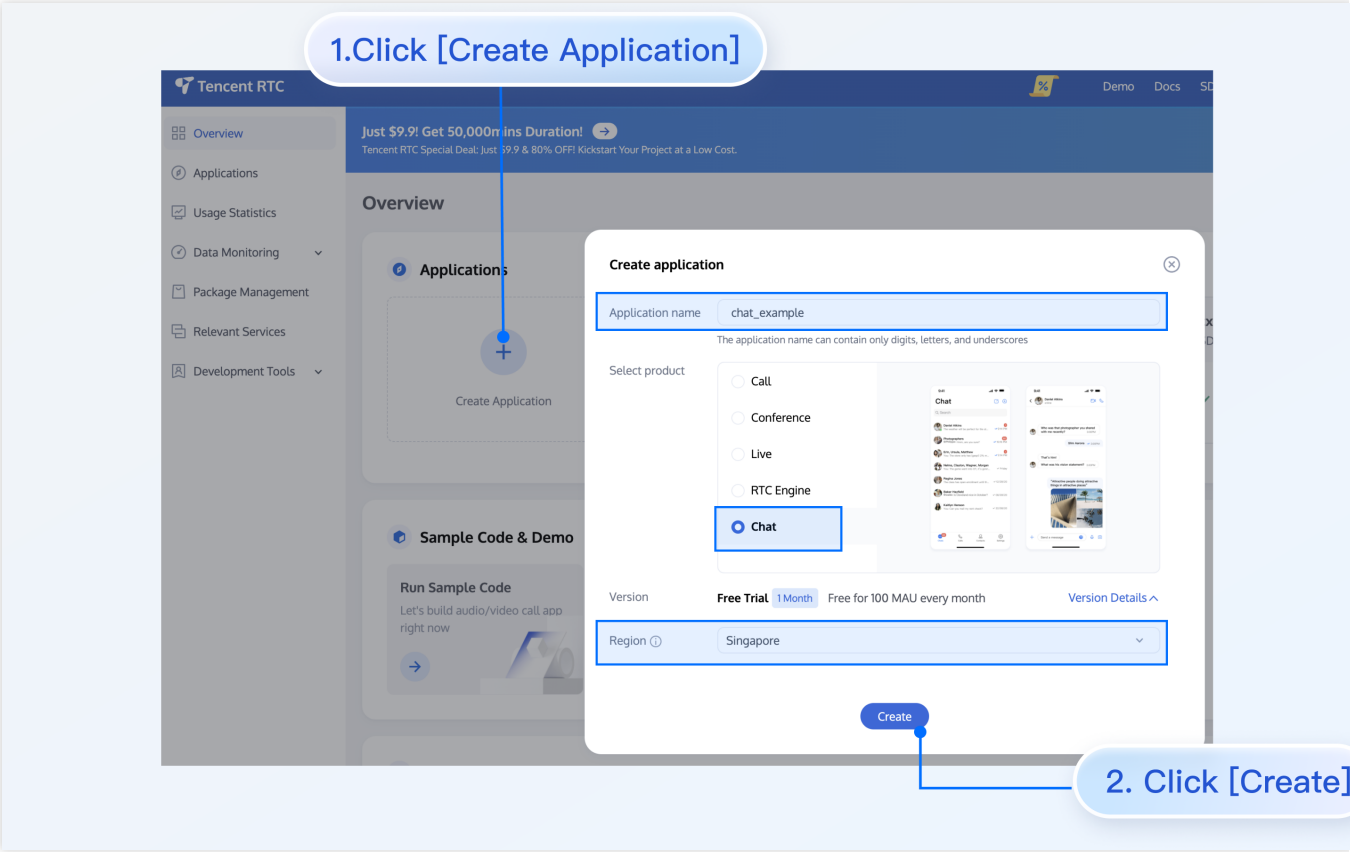
You **must** initialize the SDK before using its features. In most scenarios, you need to initialize the SDK only once during the application lifecycle.

Initialization

You can initialize the SDK in the following steps:
1. Prepare an `SDKAppID` .
2. Configure the `V2TIMSDKConfig` object.
3. Add an SDK event listener.
4. Call `initSDK` to initialize the SDK.
The detailed steps are as follows.

Preparing an SDKAppID

`SDKAppID` uniquely identifies an account. We recommend you apply for a new `SDKAppID` for each application. Messages are naturally isolated and cannot communicate between different `SDKAppID` values. To perform the initialization, you must have a correct `SDKAppID` .
In the Chat Console, you can view all your `SDKAppID` values, and you can click **Create Application** to create an `SDKAppID` .



Configuring the `V2TIMSDKConfig`

Before initializing the SDK, you need to initialize the `V2TIMSDKConfig` object (Java / Swift / Objective-C / C++), which is used to configure the SDK initially, such as setting the log level and log listening callback.

Setting the log level

The SDK supports the following log levels:

Log Level	Log Output
V2TIM_LOG_NONE	No log is output.
V2TIM_LOG_DEBUG	Logs at the DEBUG, INFO, WARNING, and ERROR levels are output (default log levels).
V2TIM_LOG_INFO	Logs at the INFO, WARNING, and ERROR levels are output.
V2TIM_LOG_WARN	Logs at the WARNING and ERROR levels are output.
V2TIM_LOG_ERROR	Logs at the ERROR level are output.

SDK log storage rules are as follows:

©2013-2025 Tencent Cloud International Pte. Ltd.

Local SDK logs are retained for seven days by default, after which the logs will be automatically cleared during the SDK initialization.
By default, SDK logs on Android are stored in the `/sdcard/tencent/imsdklogs/application package name` directory for versions earlier than v4.8.50 and in the `/sdcard/Android/data/package name/files/log/tencent/imsdk` directory for v4.8.50 or later.
By default, SDK logs on iOS are stored in the `/Library/Caches/com_tencent_imsdk_log` directory.
By default, SDK logs on Windows are stored in the program's running directory. For example, when the program is running in the `C:\\App` directory, SDK will store logs in the `C:\\App\\com_tencent_imsdk_log` directory.

Starting from v4.7.1, the xlog module from WeChat is used to output SDK logs, which are compressed by default and must be decompressed by using the Python script.
To obtain the script for decompression, click Decode Log 27 (for Python 2.7) or Decode Log 30 (for Python 3.0).
In the Windows or macOS console, you can decompress log files by running the following command. After decompression, the file names end with "xlog.log", and you can use the text editor to open these files.

```
python decode_mars_nocrypt_log_file.py imsdk_yyyyMMdd.xlog
```

Setting a log listener

If you need to listen on SDK logs in real time, you can call `setLogListener` (Java / Swift / Objective-C / C++) to set a log listener.
After it is set, the SDK will throw log information through this callback in real time.
Note:
The callback is in the main thread. Log callbacks can be quite frequent, so be careful not to synchronize too many time-consuming tasks in the callback, which may block the main thread.
The sample code of configuring `V2TIMSDKConfig` is as follows:
Java
Swift
Objective-C
C++

```
// Initialize the `config` object
V2TIMSDKConfig config = new V2TIMSDKConfig();
// Specify the log output level
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_INFO);
// Specify the log listener
config.setLogListener(new V2TIMLogListener() {
    @Override
    public void onLog(int logLevel, String logContent) {
        // `logContent` is the SDK log content
    }
```

```
});
```

```
// Initialize the `config` object
let config = V2TIMSDKConfig()
// Specify the log output level
config.logLevel = .V2TIM_LOG_INFO
// Set the log listener
config.logListener = { logLevel, logContent in
       // `logContent` is the SDK log content
    print("Log Level: \\(logLevel), Log Content: \\(logContent)")
}
```

```
// Initialize the `config` object
V2TIMSDKConfig *config = [[V2TIMSDKConfig alloc] init];
// Specify the log output level
config.logLevel = V2TIM_LOG_INFO;
// Set the log listener
config.logListener = ^(V2TIMLogLevel logLevel, NSString *logContent) {
    // `logContent` is the SDK log content
};
```

```
class LogListener final : public V2TIMLogListener {
public:
    LogListener() = default;
    ~LogListener() override = default;
    void OnLog(V2TIMLogLevel logLevel, const V2TIMString& logContent) override {
        // `logContent` is the SDK log content
    }
};
// Note that `logListener` should not be released before the SDK is uninitialized,
// otherwise the log callback cannot be called.
LogListener logListener;

// Initialize the `config` object
V2TIMSDKConfig config;
// Specify the log output level
config.logLevel = V2TIMLogLevel::V2TIM_LOG_INFO;
// Specify the log listener
config.logListener = &logListener;
```

Adding an SDK event listener

After the initialization, the SDK will report such events as connection status and login ticket expiration through `V2TIMSDKListener`.

We recommend you call the `addIMSDKListener` API (Java / Swift / Objective-C / C++) to add an SDK event listener and perform logic processing in the corresponding callback.
`V2TIMSDKListener` callbacks are as follows:

Event Callback	Description	Recommended Operation
onConnecting	The SDK is connecting to the CVM instance.	Display the "connecting" status on the UI.
onConnectSuccess	The SDK is successfully connected to the CVM instance.	-
onConnectFailed	The SDK failed to connect to the CVM instance.	Notify the user that the network connection is currently unavailable.
onKickedOffline	The current user is kicked offline.	Display the "You are already logged in on another device. Are you sure you want to log in again?" message on the UI.
onUserSigExpired	The login ticket expired.	Log in with a new `UserSig` .
onSelfInfoUpdated	The current user's profile is updated.	Update the profile photo and nickname on the UI.

Note:
If you receive the `onUserSigExpired` callback, the `UserSig` that you use for login has expired. In this case, you need to use a new `UserSig` to log in again. If you continue to use the expired `UserSig` , the SDK will be in an infinite login loop.
Sample code:
Java
Swift
Objective-C
C++

```
// The `sdkListener` type is `V2TIMSDKListener`.
V2TIMManager.getInstance().addIMSDKListener(sdkListener);
```

```
// The `self` type is id<V2TIMSDKListener>.
V2TIMManager.shared.addIMSDKListener(listener: self)
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// The `self` type is id<V2TIMSDKListener>.
[[V2TIMManager sharedInstance] addIMSDKListener:self];
```

```
// `sdkListener` is an instance of V2TIMSDKListener.
V2TIMManager::GetInstance()->AddSDKListener(&sdkListener);
```

Calling the initialization API

After performing the above steps, you can call `initSDK` (Java / Swift / Objective-C / C++) to initialize the SDK.

Sample code:

Java

Swift

Objective-C

C++

```
// 1. Get the `SDKAppID` from the IM console.
// 2. Initialize the `config` object.
V2TIMSDKConfig config = new V2TIMSDKConfig();
// 3. Specify the log output level.
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_INFO);
// 4. Add the `V2TIMSDKListener` event listener. `sdkListener` is the implementatio
V2TIMManager.getInstance().addIMSDKListener(sdkListener);
// 5. Initialize the SDK. You can call the login API as soon as you call this API.
int  sdkAppID = 1400000000; // please set the sdkAppID of your own application
V2TIMManager.getInstance().initSDK(context, sdkAppID, config);
```

```
// 1. Get the `SDKAppID` from the IM console.
// 2. Initialize the `config` object.
let config = V2TIMSDKConfig()
// 3. Specify the log output level.
config.logLevel = .V2TIM_LOG_INFO // Use enumeration values
// 4. Add the `V2TIMSDKListener` event listener. `self` is the implementation class
V2TIMManager.shared.addIMSDKListener(listener: self)
// 5. Initialize the SDK. You can call the login API as soon as you call this API.
V2TIMManager.shared.initSDK(sdkAppID: sdkAppID, config: config)
```

```
// 1. Get the `SDKAppID` from the IM console.
// 2. Initialize the `config` object.
V2TIMSDKConfig *config = [[V2TIMSDKConfig alloc] init];
// 3. Specify the log output level.
config.logLevel = V2TIM_LOG_INFO;
// 4. Add the `V2TIMSDKListener` event listener. `self` is the implementation class
```

```
[[V2TIMManager sharedInstance] addIMSDKListener:self];
// 5. Initialize the SDK. You can call the login API as soon as you call this API.
int  sdkAppID = 1400000000; // please set the sdkAppID of your own application
[[V2TIMManager sharedInstance] initSDK:sdkAppID config:config];
```

```
// 1. Get the `SDKAppID` from the IM console.
// 2. Initialize the `config` object.
V2TIMSDKConfig config;
// 3. Specify the log output level.
config.logLevel = V2TIMLogLevel::V2TIM_LOG_INFO;
// 4. Add the `V2TIMSDKListener` event listener. `sdkListener` is the implementatio
//    If you don't need to listen to SDK events, skip this step.
V2TIMManager::GetInstance()->AddSDKListener(&sdkListener);
// 5. Initialize the SDK. You can call the login API as soon as you call this API.
int  sdkAppID = 1400000000; // please set the sdkAppID of your own application
V2TIMManager::GetInstance()->InitSDK(sdkAppID, config);
```

Uninitialization

Generally, if your application's lifecycle is the same as the SDK's lifecycle, you don't need to uninitialize the SDK before exiting the application.

However, you can uninitialize the SDK in special cases, for example, only after you enter a specific UI and no longer use it after exiting the UI.

Uninitialization requires two steps:

1. If you have called `addIMSDKListener` to add the SDK listener, call `removeIMSDKListener` (Java / Swift / Objective-C / C++) to remove it.
2. Call the `unInitSDK` uninitialization API (Java / Swift / Objective-C / C++).

Sample code:

Java

Swift

Objective-C

C++

```
// Remove the `V2TIMSDKListener` event listener. `sdkListener` is the implementatio
V2TIMManager.getInstance().removeIMSDKListener(sdkListener);
// Uninitialize the SDK
V2TIMManager.getInstance().unInitSDK();
```

```
// `self` is the implementation class of id<V2TIMSDKListener>.
V2TIMManager.shared.removeIMSDKListener(listener: self)
// Uninitialize the SDK
V2TIMManager.shared.unInitSDK()
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// `self` is the implementation class of id<V2TIMSDKListener>.
[[V2TIMManager sharedInstance] removeIMSDKListener:self];
// Uninitialize the SDK
[[V2TIMManager sharedInstance] unInitSDK];
```

```
// Remove the `V2TIMSDKListener` event listener. `sdkListener` is the instance of V
V2TIMManager::GetInstance()->RemoveSDKListener(&sdkListener);
// Uninitialize the SDK
V2TIMManager::GetInstance()->UnInitSDK();
```

FAQs

1. **What should I do if error code 6013 is returned along with the description "not initialized" when I call the login or another API?**

You must initialize the SDK before using the login, message, group, conversation, relationship chain and profile, and signaling features.

©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript

Last updated：2025-01-07 15:15:29

Feature Description

You **must** initialize the Chat SDK before using its features.

Initialization

You can initialize the SDK in the following steps:

1. Prepare an `SDKAppID`.
2. Call `TencentCloudChat.create` to initialize the SDK.
3. Add SDK event listeners.

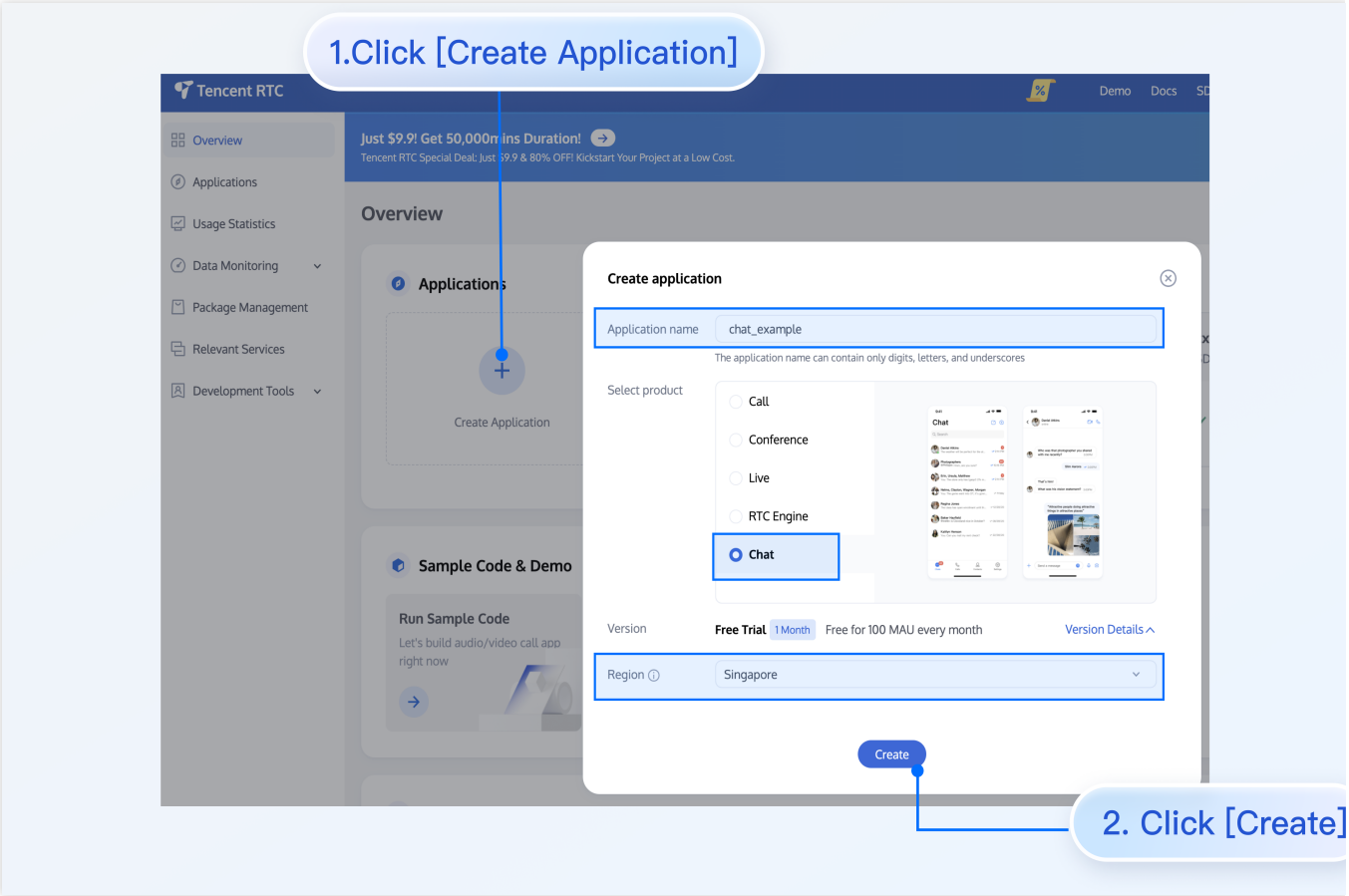
The detailed steps are as follows.

Preparing an SDKAppID

To perform the initialization, you must have a correct `SDKAppID`.

The `SDKAppID` uniquely identifies a Tencent Cloud Chat account. We recommend you apply for a new `SDKAppID` for each application. Messages are naturally isolated and cannot communicate between different `SDKAppID` values.

In the Chat Console, you can view all your `SDKAppID` values, and you can click **Create Application** to create an `SDKAppID`.



Calling the initialization API

After performing the above steps, you can call `TencentCloudChat.create` to initialize the SDK.

API

```
TencentCloudChat.create(options);
```

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
SDKAppID	Number	`SDKAppID` of the chat app
proxyServer	String | undefined	WebSocket proxy server
fileUploadProxy	String | undefined	Image, video, and file upload proxy address (the mini-program platform does not support the use of IP addresses)
fileDownloadProxy	String | undefined	Image, video, and file download proxy address (the mini-program platform does not support the use of IP addresses)

©2013-2025 Tencent Cloud International Pte. Ltd.

Examples

```
import TencentCloudChat from '@tencentcloud/chat';
import TIMUploadPlugin from 'tim-upload-plugin';

let options = {
  SDKAppID: 0 // Replace 0 with the `SDKAppID` of your chat application when connec
};
// Create an SDK instance.
// The `TencentCloudChat.create()` method returns the same instance for the same `S
// The SDK instance is usually represented by `chat`.
let chat = TencentCloudChat.create(options);

// Set the SDK log output level.
// 0 - Common level. We recommend you use this level during access as it covers mor
// 1 - Release level. We recommend you use this log level in a production environme
chat.setLogLevel(0);
// chat.setLogLevel(1);

// Register the Tencent Cloud Chat upload plugin.
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});
```

Listening on events

Note:

Please call this interface to listen for events before calling the login API to avoid missing events dispatched by the SDK.

API

```
chat.on(eventName, handler, context);
```

Name	Type	Description
eventName	String	Event names. All event names are stored in the `TencentCloudChat.EVENT` variable. If you need to view them, you can use `console.log(TencentCloudChat.EVENT)` to display all the events.
handler	Function	The method for handling events. When an event is triggered, this handler will be called to process it.
context	Object | undefined	The expected context in which the handler executes.

©2013-2025 Tencent Cloud International Pte. Ltd.

Examples

```
let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array to store Messages - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

Unlistening on events

API

```
chat.off(eventName, handler, context, once);
```

Name	Type	Description
eventName	String	Event names. All event names are stored in the `TencentCloudChat.EVENT` variable. If you need to view them, you can use `console.log(TencentCloudChat.EVENT)` to display all the events.
handler	Function	The method for handling events. When an event is triggered, this handler will be called to process it.
context	Object | undefined	The expected context in which the handler executes.
once	Boolean | undefined	Whether to unbind only once.

Examples

```
let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array to store Messages - [Message]
};
chat.off(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

List of events that the integration side needs to listen and handle

SDK_READY

This event is triggered when the SDK enters the `ready` status. When SDK is ready, you can call SDK APIs such as the message sending API to use various features of the SDK.

```
let onSdkReady = function(event) {
  let message = chat.createTextMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { text: 'Hello world!'
  }});
  chat.sendMessage(message);
};
chat.on(TencentCloudChat.EVENT.SDK_READY, onSdkReady);
```

SDK_NOT_READY

This event is triggered when the SDK enters the `not ready` status. When SDK is not ready, you cannot use SDK features such as message sending. To use them, you need to call the `login` API to drive the SDK into the `ready` status.

```
let onSdkNotReady = function(event) {
  // chat.login({userID: 'your userID', userSig: 'your userSig'});
};
chat.on(TencentCloudChat.EVENT.SDK_NOT_READY, onSdkNotReady);
```

MESSAGE_RECEIVED

This event is triggered when the SDK receives a newly pushed one-to-one message, group message, group tip, or group system message. When this event occurs, you can traverse event.data to obtain the message list and render it to the UI.

```
let onMessageReceived = function(event) {
  // event.data - An array that stores Message objects - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

MESSAGE_MODIFIED

This event is triggered when the SDK receives a notification for message modifications. When this event occurs, the message sender can traverse event.data to obtain the message list and update the content of the message with the same ID on the UI.

```
let onMessageModified = function(event) {
  // event.data - An array that stores modified Message objects - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_MODIFIED, onMessageModified);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

MESSAGE_REVOKED

This event is triggered when the SDK receives a notification indicating that messages are recalled. When this event occurs, the access side can traverse event.data to obtain data of the message recall list and render the recalled messages to the UI. For example, The peer end has recalled a message can be displayed if a message is recalled during a one-to-one conversation, and "XXX has recalled a message" can be displayed if a message is recalled during a group conversation.

```
let onMessageRevoked = function(event) {
  // event.data - An array that stores Message objects - [Message]
  // The `isRevoked` attribute value of each Message object is `true`
};
chat.on(TencentCloudChat.EVENT.MESSAGE_REVOKED, onMessageRevoked);
```

MESSAGE_READ_BY_PEER

This event is triggered when the SDK receives a notification indicating that messages have been read by peer.

```
let onMessageReadByPeer = function(event) {
  // event.data - An array that stores Message objects - [Message]
  // The `isPeerRead` attribute value of each Message object is `true`
};
chat.on(TencentCloudChat.EVENT.MESSAGE_READ_BY_PEER, onMessageReadByPeer);
```

MESSAGE_READ_RECEIPT_RECEIVED

This event is triggered when the SDK receives message read receipts.

```
let onMessageReadReceiptReceived = function(event) {
  // event.data - An array that stores read receipts
  const readReceiptInfoList = event.data;
  readReceiptInfoList.forEach((item) => {
    const { groupID, userID, messageID, readCount, unreadCount, isPeerRead } = item
    // messageID - message ID
    // userID - receiver ID
    // isPeerRead - whether the message is read by peer
    // groupID - group ID
    // readCount - count of members read the message
    // unreadCount - count of members do not read the message
    const message = chat.findMessage(messageID);
    if (message) {
     if (message.conversationType === TencentCloudChat.TYPES.CONV_C2C) {
       if (message.readReceiptInfo.isPeerRead === true) {
         // message read by peer
       }
     } else if (message.conversationType === TencentCloudChat.TYPES.CONV_GROUP) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      if (message.readReceiptInfo.unreadCount === 0) {
        // message read by all group members
      } else {
        // message.readReceiptInfo.readCount
        // If you want to find out who have read the message, please call getGroupM
      }
     }
    }
  });
}
chat.on(TencentCloudChat.EVENT.MESSAGE_READ_RECEIPT_RECEIVED, onMessageReadReceiptR
```

MESSAGE_EXTENSIONS_UPDATED

This event is triggered when the SDK receives a notification indicating that message extensions are updated.

```
let onMessageExtensionsUpdated = function(event) {
  const { messageID, extensions } = event.data;
  // messageID - message ID
  // extensions - list of message extensions
  extensions.forEach((item) => {
   const { key, value } = item;
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_EXTENSIONS_UPDATED, onMessageExtensionsUpdat
```

MESSAGE_EXTENSIONS_DELETED

This event is triggered when the SDK receives a notification indicating that message extensions are deleted.

```
let onMessageExtensionsDeleted = function(event) {
  const { messageID, keyList } = event.data;
  // messageID - message ID
  // keyList - list of keys which are deleted
  keyList.forEach((key) => {
   // console.log(key)
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_EXTENSIONS_DELETED, onMessageExtensionsDelet
```

CONVERSATION_LIST_UPDATED

This event is triggered when the conversation list is updated. event.data is an array that stores the Conversation objects.

```
let onConversationListUpdated = function(event) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  console.log(event.data); // Array that stores Conversation objects - [Conversatio
};
chat.on(TencentCloudChat.EVENT.CONVERSATION_LIST_UPDATED, onConversationListUpdated
```

TOTAL_UNREAD_MESSAGE_COUNT_UPDATED

This event is triggered when the total unread message count of all conversations updated.

```
let onTotalUnreadMessageCountUpdated = function(event) {
  console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOTAL_UNREAD_MESSAGE_COUNT_UPDATED, onTotalUnreadMes
```

CONVERSATION_GROUP_LIST_UPDATED

This event is triggered when the conversation group list is updated.

```
let onConversationGroupListUpdated = function(event) {
  console.log(event.data);
}
chat.on(TencentCloudChat.EVENT.CONVERSATION_GROUP_LIST_UPDATED, onConversationGroup
```

CONVERSATION_IN_GROUP_UPDATED

This event is triggered when the conversations in a conversation group list updated.

```
let onConversationInGroupUpdated = function(event) {
  const { groupName, conversationList }  = event.data;
}
chat.on(TencentCloudChat.EVENT.CONVERSATION_IN_GROUP_UPDATED, onConversationInGroup
```

GROUP_LIST_UPDATED

This event is triggered when the SDK group list updated. The access side can traverse event.data to obtain the group list and render it to the UI.

```
let onGroupListUpdated = function(event) {
   console.log(event.data);// Array that stores Group objects - [Group]
};
chat.on(TencentCloudChat.EVENT.GROUP_LIST_UPDATED, onGroupListUpdated);
```

GROUP_ATTRIBUTES_UPDATED

This event is triggered when attributes of a group updated.

```
let onGroupAttributesUpdated = function(event) {
   const groupID = event.data.groupID
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    const groupAttributes = event.data.groupAttributes
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.GROUP_ATTRIBUTES_UPDATED, onGroupAttributesUpdated);
```

GROUP_COUNTER_UPDATED

This event is triggered when counters of a group updated.

```
let onGroupCounterUpdated = function(event) {
  const { groupID, key, value } = event.data;
};
chat.on(TencentCloudChat.EVENT.GROUP_COUNTER_UPDATED, onGroupCounterUpdated);
```

TOPIC_CREATED

This event is triggered when a topic created in a community group.

```
let onTopicCreated = function(event) {
    const groupID = event.data.groupID; // community group ID
    const topicID = event.data.topicID; // topic ID
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOPIC_CREATED, onTopicCreated);
```

TOPIC_DELETED

This event is triggered when a topic removed from a community group.

```
let onTopicDeleted = function(event) {
    const groupID = event.data.groupID; // community group ID
    const topicIDList = event.data.topicIDList; // topic ID
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOPIC_DELETED, onTopicDeleted);
```

TOPIC_UPDATED

This event is triggered when a topic updated in a community group.

```
let onTopicUpdated = function(event) {
    const groupID = event.data.groupID; // community group ID
    const topic = event.data.topic; // the lastest topic
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOPIC_UPDATED, onTopicUpdated);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

PROFILE_UPDATED

This event is triggered when the profile of the current user or profiles of friends are changed. `event.data` is an array that stores Profile objects.

```
let onProfileUpdated = function(event) {
  console.log(event.data); // Array that stores Profile objects
};
chat.on(TencentCloudChat.EVENT.PROFILE_UPDATED, onProfileUpdated);
```

USER_STATUS_UPDATED

This event is triggered when statuses of subscribed users or statuses of friends updated(Including online status and custom status).

```
let onUserStatusUpdated = function(event) {
   console.log(event.data);
   const userStatusList = event.data;
   userStatusList.forEach((item) => {
     const { userID, statusType, customStatus } = item;
     // userID
     // statusType, described as follows:
     // TencentCloudChat.TYPES.USER_STATUS_UNKNOWN
     // TencentCloudChat.TYPES.USER_STATUS_ONLINE
     // TencentCloudChat.TYPES.USER_STATUS_OFFLINE
     // TencentCloudChat.TYPES.USER_STATUS_UNLOGINED
     // customStatus
   })
};
chat.on(TencentCloudChat.EVENT.USER_STATUS_UPDATED, onUserStatusUpdated);
```

BLACKLIST_UPDATED

This event is triggered when the SDK blocklist is updated.

```
let onBlacklistUpdated = function(event) {
  console.log(event.data); // Your blocklist. The value is an array that stores `us
};
chat.on(TencentCloudChat.EVENT.BLACKLIST_UPDATED, onBlacklistUpdated);
```

FRIEND_LIST_UPDATED

This event is triggered when the friend list is updated.

```
let onFriendListUpdated = function(event) {
  console.log(event.data);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
chat.on(TencentCloudChat.EVENT.FRIEND_LIST_UPDATED, onFriendListUpdated);
```

FRIEND_GROUP_LIST_UPDATED

This event is triggered when the friend group list is updated.

```
let onFriendGroupListUpdated = function(event) {
  console.log(event.data);
}
chat.on(TencentCloudChat.EVENT.FRIEND_GROUP_LIST_UPDATED, onFriendGroupListUpdated)
```

FRIEND_APPLICATION_LIST_UPDATED

This event is triggered when the friend application list is updated.

```
let onFriendApplicationListUpdated = function(event) {
  // friendApplicationList - Friend request list - [FriendApplication]
  // unreadCount - Number of unread friend requests
  const { friendApplicationList, unreadCount } = event.data;
  // Friend requests received by you (friend requests that are sent to you by other
  const applicationSentToMe = friendApplicationList.filter((friendApplication) => f
  // Friend requests sent by you (friend requests that you send to others)
  const applicationSentByMe = friendApplicationList.filter((friendApplication) => f
};
chat.on(TencentCloudChat.EVENT.FRIEND_APPLICATION_LIST_UPDATED, onFriendApplication
```

KICKED_OUT

This event is triggered when the current user is kicked offline.

```
let onKickedOut = function(event) {
  console.log(event.data.type);
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_ACCOUNT
  // The user is forcibly logged out because the same account logs in from multiple
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_DEVICE
  // The user is forcibly logged out because the same account logs in from multiple
  // TencentCloudChat.TYPES.KICKED_OUT_USERSIG_EXPIRED
 // The signature expired
  // TencentCloudChat.TYPES.KICKED_OUT_REST_API
 // The user is forcibly logged out by the RESTful API.
};
chat.on(TencentCloudChat.EVENT.KICKED_OUT, onKickedOut);
```

NET_STATE_CHANGE

This event is triggered when the network status changes.

©2013-2025 Tencent Cloud International Pte. Ltd.

```
let onNetStateChange = function(event) {
  console.log(event.data.type);
  // TencentCloudChat.TYPES.NET_STATE_CONNECTED - Already connected to the network
  // TencentCloudChat.TYPES.NET_STATE_CONNECTING
  // Connecting. This often occurs when the SDK must reconnect due to network jitte
  // "The current network is unstable" or "Connecting..." can be displayed at the a
  // TencentCloudChat.TYPES.NET_STATE_DISCONNECTED
  // Disconnected. "The current network is unavailable" can be displayed at the acc
  // The SDK will continue to try to reconnect. If the user network recovers, the S
};
chat.on(TencentCloudChat.EVENT.NET_STATE_CHANGE, onNetStateChange);
```

Deinitialization

Destroy the SDK instance. The SDK will first log out, then disconnect the WebSocket long connection, and release resources.

```
chat.destroy();
```

Next step, login and logout.

Flutter

Last updated：2025-04-16 14:28:45

Feature Description

You **must** initialize the Chat SDK before using its features.
In most scenarios, you need to initialize the Chat SDK only once during the application lifecycle.

Initialization

You can initialize the SDK in the following steps:
1. Prepare an `SDKAppID`.
2. Set the `LogLevelEnum`.
3. Set the SDK event listener.
4. Call `initSDK` to initialize the SDK.
The detailed steps are as follows.

Preparing an SDKAppID

To perform the initialization, you must have a correct `SDKAppID`.
`SDKAppID` is the unique ID that the Chat service uses to identify a customer account. We recommend you apply for a new `SDKAppID` for every independent app to automatically isolate messages between `SDKAppIDs`.
You can view all `SDKAppIDs` in the Chat console or click **Create Application** to create an `SDKAppID`.

Setting the LogLevelEnum

Before initializing the SDK, you need to initialize the `LogLevelEnum` (Dart) object, which is used to set the SDK log level.

Setting the log level

The Chat SDK supports the following log levels:

Log Level	Log Output
LogLevelEnum.V2TIM_LOG_NONE	No log is output.
LogLevelEnum.V2TIM_LOG_DEBUG	Logs of the DEBUG, INFO, WARNING, and ERROR levels (default

©2013-2025 Tencent Cloud International Pte. Ltd.

	log levels) are output.
LogLevelEnum.V2TIM_LOG_INFO	Logs at the INFO, WARNING, and ERROR levels are output.
LogLevelEnum.V2TIM_LOG_WARN	Logs at the WARNING and ERROR levels are output.
LogLevelEnum.V2TIM_LOG_ERROR	Logs at the ERROR level are output.

SDK log storage rules are as follows:
Local Chat SDK logs are stored for seven days by default, and logs generated earlier than seven days ago will be automatically cleared during SDK initialization.
For Android, Chat SDK logs are stored in the `/sdcard/tencenet/imsdklogs/<App package name>` directory by default for versions earlier than 4.8.50 and in the `/sdcard/Android/data/<Package name>/files/log/tencent/imsdk` directory for version 4.8.50 or later.
Starting from v4.7.1, the xlog module from WeChat is used to output Chat SDK logs, which are compressed by default and must be decompressed by using the Python script.
To obtain the script for decompression, click Decode Log 27 (for Python 2.7) or Decode Log 30 (for Python 3.0).
In the Windows or Mac console, you can decompress log files by running the following command. After decompression, the file names end with "xlog.log", and you can use the text editor to open these files.

```
python decode_mars_nocrypt_log_file.py imsdk_yyyyMMdd.xlog
```

Setting the SDK event listener

After the initialization, the SDK will report such events as connection status and login ticket expiration through `V2TimSDKListener`.
We recommend you pass in `V2TimSDKListener` (Dart) when calling `initSDK` to add the SDK event listener and perform logic processing in the callback.
`V2TimSDKListener` callbacks are as follows:

Event Callback	Event Description	Recommended Operation
onConnecting	The SDK is connecting to the CVM instance.	Display the "connecting" status on the UI.
onConnectSuccess	The SDK is successfully connected to the CVM instance.	-
onConnectFailed	The SDK failed to connect to the CVM instance.	Notify the user that the network connection is currently unavailable.
onKickedOffline	The current user is kicked	Display the "You are already logged in on another device.

	offline.	Are you sure you want to log in again?" message on the UI.
onUserSigExpired	The login ticket expired.	Log in with a new `UserSig` .
onSelfInfoUpdated	The current user's profile is updated.	Update the profile photo and nickname on the UI.

Caution:

If you receive the `onUserSigExpired` callback, the UserSig that you use for login has expired. In this case, you need to use the newly issued `UserSig` to log in again. If you continue to use the expired `UserSig` , the Chat SDK will enter an infinite login loop.

Calling the initialization API

After performing the above steps, you can call `initSDK` (Dart) to initialize the SDK.

Sample code:

```
// 1. Get the `SDKAppID` from the IM console.
int sdkAppID = 0;
// 2. Add the `V2TimSDKListener` event listener. `sdkListener` is the implement
V2TimSDKListener sdkListener = V2TimSDKListener(
  onConnectFailed: (int code, String error) {
    // Connection failure callback function
    // `code`: Error code
    // `error`: Error message
  },
  onConnectSuccess: () {
    // The SDK is successfully connected to the CVM instance
  },
  onConnecting: () {
    // The SDK is connecting to the CVM instance
  },
  onKickedOffline: () {
    // The current user is kicked offline: the SDK notifies the user on the UI,
  },
  onSelfInfoUpdated: (V2TimUserFullInfo info) {
    // The profile of the current user was updated
    // `info`: information of the login user
  },
  onUserSigExpired: () {
    // The ticket expires when the user is online: the user needs to generate a
  },
  onUserStatusChanged: (List<V2TimUserStatus> userStatusList) {
    // User status change notification
    // `userStatusList`: list of users whose status changes
```

```
        // Notification receiving: This callback will be triggered if a subscribed
        // This callback will be triggered when a friend's status changes after not
        // This callback will be sent to all the devices when an account is logged
      },
    );
    // 3. Initialize the SDK
    V2TimValueCallback<bool> initSDKRes =
        await TencentImSDKPlugin.v2TIMManager.initSDK(
      sdkAppID: sdkAppID, // SDKAppID
      loglevel: LogLevelEnum.V2TIM_LOG_ALL, // Log registration level
      listener: sdkListener, // Event listener
    );
    if (initSDKRes.code == 0) {
      // Initialized successfully
    }
```

// Uninitialization

Generally, if your application's lifecycle is the same as the Chat SDK's lifecycle, you don't need to uninitialize the Chat SDK before exiting the application.

However, you can uninitialize the Chat SDK in special cases, for example, only after you enter a specific UI and no longer use it after exiting the UI.

You can perform the uninitialization by calling the uninitialization API `unInitSDK` (Dart).

Sample code:

```
// Uninitialize the SDK
TencentImSDKPlugin.v2TIMManager.unInitSDK();
```

FAQs

1. What should I do if error code 6013 is returned along with the description "not initialized" when I call the login or another API?

You must initialize the Chat SDK before using the login, message, group, conversation, relationship chain and profile, and signaling features.

Unity

Last updated：2025-03-20 16:42:06

Overview

You **must** initialize the Chat SDK before using its features.
In most scenarios, you need to initialize the Chat SDK only once during the application lifecycle.

Initialization

You can initialize the SDK in the following steps:
1. Prepare an `SDKAppID`.
2. Set the `SdkConfig`.
3. Set the SDK event listener.
4. Call `Init` to initialize the SDK.
The detailed steps are as follows.

Preparing an SDKAppID

To perform the initialization, you must have a correct `SDKAppID`.
`SDKAppID` is the unique ID that the Chat service uses to identify a customer account. We recommend you apply for a new `SDKAppID` for every independent app to automatically isolate messages between `SDKAppIDs`.
You can view all `SDKAppIDs` in the Chat console or click **Create Application** to create an `SDKAppID`.

Setting the SdkConfig

Before initializing the SDK, you need to initialize the SdkConfig object, which is used to set the local SDK cache and log position.
It is used to configure the storage path of Chat running logs and data.

sdk_config_config_file_path

Storage path of local Chat data.

Caution

The app needs to have read-write access to this path.

©2013-2025 Tencent Cloud International Pte. Ltd.

sdk_config_log_file_path

It is the storage path of the Chat logs.

Caution

The app needs to have read-write access to this path.

Calling the initialization API

After performing the above steps, you can call `Init` to initialize the SDK.

Sample code:

```
namespace Com.Tencent.IM.Unity.UIKit{
    public static void Init() {
        string sdkappid = ""; // Get the `SDKAppID` from the IM console
        SdkConfig sdkConfig = new SdkConfig();

        sdkConfig.sdk_config_config_file_path = Application.persistentDataPath + "/

        sdkConfig.sdk_config_log_file_path = Application.persistentDataPath + "/TIM

        TIMResult res = TencentIMSDK.Init(long.Parse(sdkappid), sdkConfig);
    }
}
```

Registering a SDK global event listener

After the initialization, SDK will throw such events as connection status and login ticket expiration through the `NetworkStatusListenerCallback`, `UserSigExpiredCallback`, and other callbacks.

We recommend you register a global event listener immediately after calling `initSDK` and perform logic processing in such callbacks.

The callbacks are as described below:

Event Callback	Description
RecvNewMsgCallback	Callback for receiving a new message
MsgReadedReceiptCallback	Callback for a message read receipt
MsgRevokeCallback	Callback for a message recall
MsgElemUploadProgressCallback	Callback for the upload progress of a message element
GroupTipsEventCallback	Callback for a group system message
GroupAttributeChangedCallback	Callback for a group attribute change
ConvTotalUnreadMessageCountChangedCallback	Callback for a change in the unread message count of a

	conversation
NetworkStatusListenerCallback	Callback for listening for the network connection status
KickedOfflineCallback	Callback for being kicked offline
UserSigExpiredCallback	Callback for ticket expiration
OnAddFriendCallback	Callback for adding a friend
OnDeleteFriendCallback	Callback for deleting a friend
UpdateFriendProfileCallback	Callback for updating the profile of a friend
FriendAddRequestCallback	Callback for a friend request
FriendApplicationListDeletedCallback	Callback for deleting a friend request
FriendApplicationListReadCallback	Callback for reading a friend request
FriendBlackListAddedCallback	Callback for adding a friend to the blocklist
FriendBlackListDeletedCallback	Callback for deleting a friend from the blocklist
LogCallback	Log callback
MsgUpdateCallback	Callback for a message update
MsgGroupMessageReadMemberListCallback	Callback for getting the list of group members who have read a group message

Caution

If you receive the `UserSigExpiredCallback` callback, the `UserSig` that you use for login has expired. In this case, you need to use the newly issued `UserSig` to log in again. If you continue to use the expired `UserSig`, the Chat SDK will enter an infinite login loop.

// Uninitialization

Generally, if your application's lifecycle is the same as the Chat SDK's lifecycle, you don't need to uninitialize the Chat SDK before exiting the application.

However, you can uninitialize the Chat SDK in special cases, for example, only after you enter a specific UI and no longer use it after exiting the UI.

You can perform the uninitialization by calling the uninitialization API `unInit`.

Sample code:

```
// Uninitialize the SDK
TencentIMSDK.Uninit();
```

Others

It is used to display the result returned when the SDK is called. When `res` is `TIMResult.TIM_SUCC = 0`, the API call is successful.
After the SDK is successfully initialized, add required listeners to avoid missing messages.

FAQs

1. You must initialize the Chat SDK before using the login, message, group, conversation, relationship chain and profile, and signaling features.

React Native

Last updated：2025-01-07 15:15:30

Feature Description

You **must** initialize the Chat SDK before using its features.

Initialization

You can initialize the SDK in the following steps:

1. Prepare an `SDKAppID`.
2. Call `TencentCloudChat.create` to initialize the SDK.
3. Add SDK event listeners.

The detailed steps are as follows.

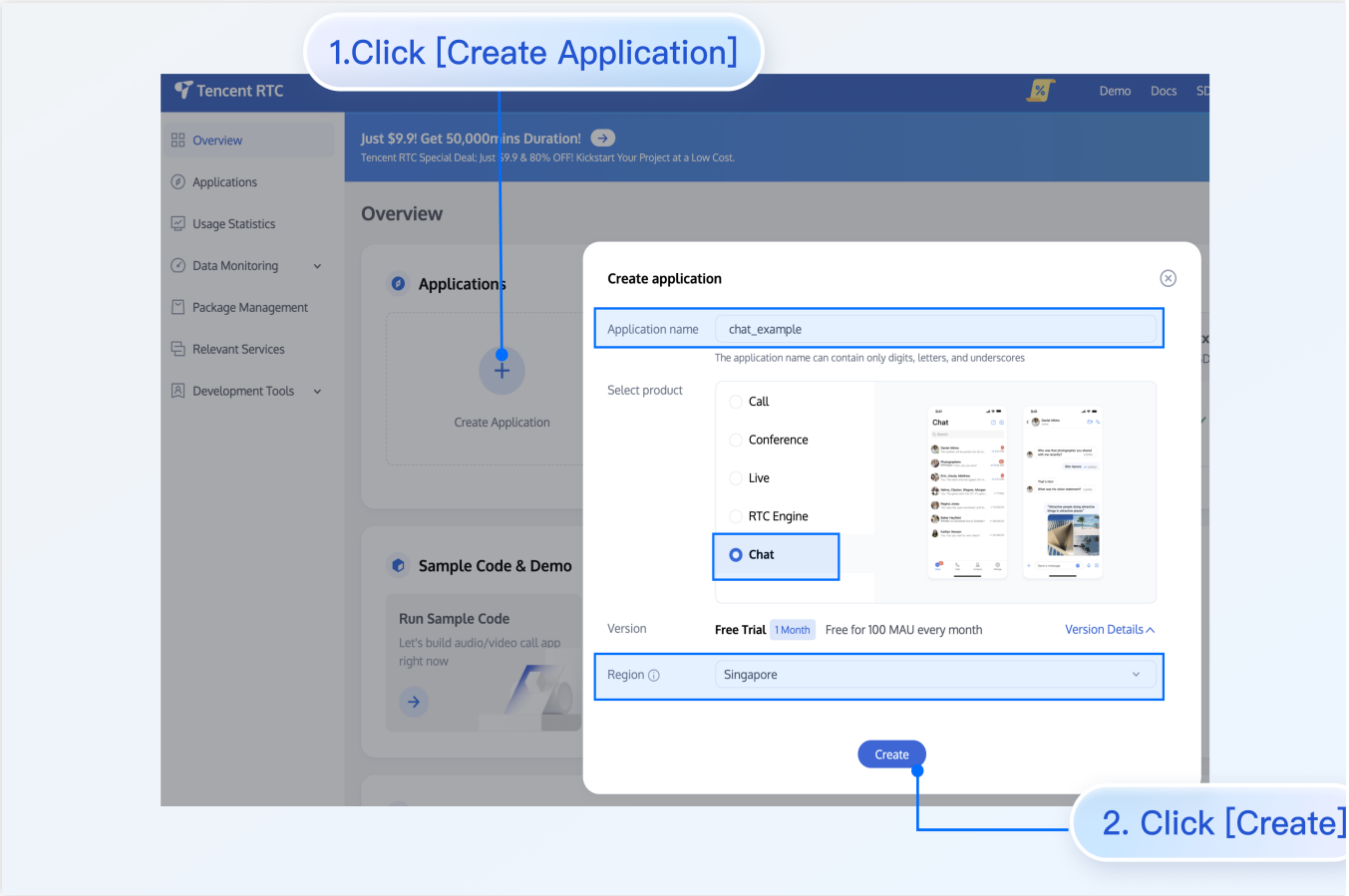
Preparing an SDKAppID

To perform the initialization, you must have a correct `SDKAppID`.

The `SDKAppID` uniquely identifies a Tencent Cloud Chat account. We recommend you apply for a new `SDKAppID` for each application. Messages are naturally isolated and cannot communicate between different `SDKAppID` values.

In the Chat Console, you can view all your `SDKAppID` values, and you can click **Create Application** to create an `SDKAppID`.

©2013-2025 Tencent Cloud International Pte. Ltd.



Calling the initialization API

After performing the above steps, you can call `TencentCloudChat.create` to initialize the SDK.

API

```
TencentCloudChat.create(options);
```

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
SDKAppID	Number	`SDKAppID` of the chat app
proxyServer	String | undefined	WebSocket proxy server
fileUploadProxy	String | undefined	Image, video, and file upload proxy address (the mini-program platform does not support the use of IP addresses)
fileDownloadProxy	String | undefined	Image, video, and file download proxy address (the mini-program platform does not support the use of IP addresses)

©2013-2025 Tencent Cloud International Pte. Ltd.

Examples

```
import TencentCloudChat from '@tencentcloud/chat';
import TIMUploadPlugin from 'tim-upload-plugin';

let options = {
  SDKAppID: 0 // Replace 0 with the `SDKAppID` of your chat application when connec
};
// Create an SDK instance.
// The `TencentCloudChat.create()` method returns the same instance for the same `S
// The SDK instance is usually represented by `chat`.
let chat = TencentCloudChat.create(options);

// Set the SDK log output level.
// 0 - Common level. We recommend you use this level during access as it covers mor
// 1 - Release level. We recommend you use this log level in a production environme
chat.setLogLevel(0);
// chat.setLogLevel(1);

// Register the Tencent Cloud Chat upload plugin.
chat.registerPlugin({'tim-upload-plugin': TIMUploadPlugin});
```

Listening on events

Note:

Please call this interface to listen for events before calling the login API to avoid missing events dispatched by the SDK.

API

```
chat.on(eventName, handler, context);
```

Name	Type	Description
eventName	String	Event names. All event names are stored in the `TencentCloudChat.EVENT` variable. If you need to view them, you can use `console.log(TencentCloudChat.EVENT)` to display all the events.
handler	Function	The method for handling events. When an event is triggered, this handler will be called to process it.
context	Object | undefined	The expected context in which the handler executes.

©2013-2025 Tencent Cloud International Pte. Ltd.

Examples

```
let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array to store Messages - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

Unlistening on events

API

```
chat.off(eventName, handler, context, once);
```

Name	Type	Description
eventName	String	Event names. All event names are stored in the `TencentCloudChat.EVENT` variable. If you need to view them, you can use `console.log(TencentCloudChat.EVENT)` to display all the events.
handler	Function	The method for handling events. When an event is triggered, this handler will be called to process it.
context	Object | undefined	The expected context in which the handler executes.
once	Boolean | undefined	Whether to unbind only once.

Examples

```
let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array to store Messages - [Message]
};
chat.off(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

List of events that the integration side needs to listen and handle

SDK_READY

This event is triggered when the SDK enters the `ready` status. When SDK is ready, you can call SDK APIs such as the message sending API to use various features of the SDK.

```
let onSdkReady = function(event) {
  let message = chat.createTextMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { text: 'Hello world!'
  }});
  chat.sendMessage(message);
};
chat.on(TencentCloudChat.EVENT.SDK_READY, onSdkReady);
```

SDK_NOT_READY

This event is triggered when the SDK enters the `not ready` status. When SDK is not ready, you cannot use SDK features such as message sending. To use them, you need to call the `login` API to drive the SDK into the `ready` status.

```
let onSdkNotReady = function(event) {
  // chat.login({userID: 'your userID', userSig: 'your userSig'});
};
chat.on(TencentCloudChat.EVENT.SDK_NOT_READY, onSdkNotReady);
```

MESSAGE_RECEIVED

This event is triggered when the SDK receives a newly pushed one-to-one message, group message, group tip, or group system message. When this event occurs, you can traverse event.data to obtain the message list and render it to the UI.

```
let onMessageReceived = function(event) {
  // event.data - An array that stores Message objects - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

MESSAGE_MODIFIED

This event is triggered when the SDK receives a notification for message modifications. When this event occurs, the message sender can traverse event.data to obtain the message list and update the content of the message with the same ID on the UI.

```
let onMessageModified = function(event) {
  // event.data - An array that stores modified Message objects - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_MODIFIED, onMessageModified);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

MESSAGE_REVOKED

This event is triggered when the SDK receives a notification indicating that messages are recalled. When this event occurs, the access side can traverse event.data to obtain data of the message recall list and render the recalled messages to the UI. For example, The peer end has recalled a message can be displayed if a message is recalled during a one-to-one conversation, and "XXX has recalled a message" can be displayed if a message is recalled during a group conversation.

```
let onMessageRevoked = function(event) {
  // event.data - An array that stores Message objects - [Message]
  // The `isRevoked` attribute value of each Message object is `true`
};
chat.on(TencentCloudChat.EVENT.MESSAGE_REVOKED, onMessageRevoked);
```

MESSAGE_READ_BY_PEER

This event is triggered when the SDK receives a notification indicating that messages have been read by peer.

```
let onMessageReadByPeer = function(event) {
  // event.data - An array that stores Message objects - [Message]
  // The `isPeerRead` attribute value of each Message object is `true`
};
chat.on(TencentCloudChat.EVENT.MESSAGE_READ_BY_PEER, onMessageReadByPeer);
```

MESSAGE_READ_RECEIPT_RECEIVED

This event is triggered when the SDK receives message read receipts.

```
let onMessageReadReceiptReceived = function(event) {
  // event.data - An array that stores read receipts
  const readReceiptInfoList = event.data;
  readReceiptInfoList.forEach((item) => {
    const { groupID, userID, messageID, readCount, unreadCount, isPeerRead } = item
    // messageID - message ID
    // userID - receiver ID
    // isPeerRead - whether the message is read by peer
    // groupID - group ID
    // readCount - count of members read the message
    // unreadCount - count of members do not read the message
    const message = chat.findMessage(messageID);
    if (message) {
     if (message.conversationType === TencentCloudChat.TYPES.CONV_C2C) {
       if (message.readReceiptInfo.isPeerRead === true) {
         // message read by peer
       }
     } else if (message.conversationType === TencentCloudChat.TYPES.CONV_GROUP) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      if (message.readReceiptInfo.unreadCount === 0) {
        // message read by all group members
      } else {
        // message.readReceiptInfo.readCount
        // If you want to find out who have read the message, please call getGroupM
      }
     }
    }
  });
}
chat.on(TencentCloudChat.EVENT.MESSAGE_READ_RECEIPT_RECEIVED, onMessageReadReceiptR
```

MESSAGE_EXTENSIONS_UPDATED

This event is triggered when the SDK receives a notification indicating that message extensions are updated.

```
let onMessageExtensionsUpdated = function(event) {
  const { messageID, extensions } = event.data;
  // messageID - message ID
  // extensions - list of message extensions
  extensions.forEach((item) => {
   const { key, value } = item;
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_EXTENSIONS_UPDATED, onMessageExtensionsUpdat
```

MESSAGE_EXTENSIONS_DELETED

This event is triggered when the SDK receives a notification indicating that message extensions are deleted.

```
let onMessageExtensionsDeleted = function(event) {
  const { messageID, keyList } = event.data;
  // messageID - message ID
  // keyList - list of keys which are deleted
  keyList.forEach((key) => {
   // console.log(key)
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_EXTENSIONS_DELETED, onMessageExtensionsDelet
```

CONVERSATION_LIST_UPDATED

This event is triggered when the conversation list is updated. event.data is an array that stores the Conversation objects.

```
let onConversationListUpdated = function(event) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  console.log(event.data); // Array that stores Conversation objects - [Conversatio
};
chat.on(TencentCloudChat.EVENT.CONVERSATION_LIST_UPDATED, onConversationListUpdated
```

TOTAL_UNREAD_MESSAGE_COUNT_UPDATED

This event is triggered when the total unread message count of all conversations updated.

```
let onTotalUnreadMessageCountUpdated = function(event) {
  console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOTAL_UNREAD_MESSAGE_COUNT_UPDATED, onTotalUnreadMes
```

CONVERSATION_GROUP_LIST_UPDATED

This event is triggered when the conversation group list is updated.

```
let onConversationGroupListUpdated = function(event) {
  console.log(event.data);
}
chat.on(TencentCloudChat.EVENT.CONVERSATION_GROUP_LIST_UPDATED, onConversationGroup
```

CONVERSATION_IN_GROUP_UPDATED

This event is triggered when the conversations in a conversation group list updated.

```
let onConversationInGroupUpdated = function(event) {
  const { groupName, conversationList }  = event.data;
}
chat.on(TencentCloudChat.EVENT.CONVERSATION_IN_GROUP_UPDATED, onConversationInGroup
```

GROUP_LIST_UPDATED

This event is triggered when the SDK group list updated. The access side can traverse event.data to obtain the group list and render it to the UI.

```
let onGroupListUpdated = function(event) {
   console.log(event.data);// Array that stores Group objects - [Group]
};
chat.on(TencentCloudChat.EVENT.GROUP_LIST_UPDATED, onGroupListUpdated);
```

GROUP_ATTRIBUTES_UPDATED

This event is triggered when attributes of a group updated.

```
let onGroupAttributesUpdated = function(event) {
   const groupID = event.data.groupID
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    const groupAttributes = event.data.groupAttributes
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.GROUP_ATTRIBUTES_UPDATED, onGroupAttributesUpdated);
```

GROUP_COUNTER_UPDATED

This event is triggered when counters of a group updated.

```
let onGroupCounterUpdated = function(event) {
  const { groupID, key, value } = event.data;
};
chat.on(TencentCloudChat.EVENT.GROUP_COUNTER_UPDATED, onGroupCounterUpdated);
```

TOPIC_CREATED

This event is triggered when a topic created in a community group.

```
let onTopicCreated = function(event) {
    const groupID = event.data.groupID; // community group ID
    const topicID = event.data.topicID; // topic ID
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOPIC_CREATED, onTopicCreated);
```

TOPIC_DELETED

This event is triggered when a topic removed from a community group.

```
let onTopicDeleted = function(event) {
    const groupID = event.data.groupID; // community group ID
    const topicIDList = event.data.topicIDList; // topic ID
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOPIC_DELETED, onTopicDeleted);
```

TOPIC_UPDATED

This event is triggered when a topic updated in a community group.

```
let onTopicUpdated = function(event) {
    const groupID = event.data.groupID; // community group ID
    const topic = event.data.topic; // the lastest topic
    console.log(event.data);
};
chat.on(TencentCloudChat.EVENT.TOPIC_UPDATED, onTopicUpdated);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

PROFILE_UPDATED

This event is triggered when the profile of the current user or profiles of friends are changed. `event.data` is an array that stores Profile objects.

```
let onProfileUpdated = function(event) {
  console.log(event.data); // Array that stores Profile objects
};
chat.on(TencentCloudChat.EVENT.PROFILE_UPDATED, onProfileUpdated);
```

USER_STATUS_UPDATED

This event is triggered when statuses of subscribed users or statuses of friends updated(Including online status and custom status).

```
let onUserStatusUpdated = function(event) {
   console.log(event.data);
   const userStatusList = event.data;
   userStatusList.forEach((item) => {
     const { userID, statusType, customStatus } = item;
     // userID
     // statusType, described as follows:
     // TencentCloudChat.TYPES.USER_STATUS_UNKNOWN
     // TencentCloudChat.TYPES.USER_STATUS_ONLINE
     // TencentCloudChat.TYPES.USER_STATUS_OFFLINE
     // TencentCloudChat.TYPES.USER_STATUS_UNLOGINED
     // customStatus
   })
};
chat.on(TencentCloudChat.EVENT.USER_STATUS_UPDATED, onUserStatusUpdated);
```

BLACKLIST_UPDATED

This event is triggered when the SDK blocklist is updated.

```
let onBlacklistUpdated = function(event) {
  console.log(event.data); // Your blocklist. The value is an array that stores `us
};
chat.on(TencentCloudChat.EVENT.BLACKLIST_UPDATED, onBlacklistUpdated);
```

FRIEND_LIST_UPDATED

This event is triggered when the friend list is updated.

```
let onFriendListUpdated = function(event) {
  console.log(event.data);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
}
chat.on(TencentCloudChat.EVENT.FRIEND_LIST_UPDATED, onFriendListUpdated);
```

FRIEND_GROUP_LIST_UPDATED

This event is triggered when the friend group list is updated.

```
let onFriendGroupListUpdated = function(event) {
  console.log(event.data);
}
chat.on(TencentCloudChat.EVENT.FRIEND_GROUP_LIST_UPDATED, onFriendGroupListUpdated)
```

FRIEND_APPLICATION_LIST_UPDATED

This event is triggered when the friend application list is updated.

```
let onFriendApplicationListUpdated = function(event) {
  // friendApplicationList - Friend request list - [FriendApplication]
  // unreadCount - Number of unread friend requests
  const { friendApplicationList, unreadCount } = event.data;
  // Friend requests received by you (friend requests that are sent to you by other
  const applicationSentToMe = friendApplicationList.filter((friendApplication) => f
  // Friend requests sent by you (friend requests that you send to others)
  const applicationSentByMe = friendApplicationList.filter((friendApplication) => f
};
chat.on(TencentCloudChat.EVENT.FRIEND_APPLICATION_LIST_UPDATED, onFriendApplication
```

KICKED_OUT

This event is triggered when the current user is kicked offline.

```
let onKickedOut = function(event) {
  console.log(event.data.type);
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_ACCOUNT
  // The user is forcibly logged out because the same account logs in from multiple
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_DEVICE
  // The user is forcibly logged out because the same account logs in from multiple
  // TencentCloudChat.TYPES.KICKED_OUT_USERSIG_EXPIRED
 // The signature expired
  // TencentCloudChat.TYPES.KICKED_OUT_REST_API
 // The user is forcibly logged out by the RESTful API.
};
chat.on(TencentCloudChat.EVENT.KICKED_OUT, onKickedOut);
```

NET_STATE_CHANGE

This event is triggered when the network status changes.

```
let onNetStateChange = function(event) {
  console.log(event.data.type);
  // TencentCloudChat.TYPES.NET_STATE_CONNECTED - Already connected to the network
  // TencentCloudChat.TYPES.NET_STATE_CONNECTING
  // Connecting. This often occurs when the SDK must reconnect due to network jitte
  // "The current network is unstable" or "Connecting..." can be displayed at the a
  // TencentCloudChat.TYPES.NET_STATE_DISCONNECTED
  // Disconnected. "The current network is unavailable" can be displayed at the acc
  // The SDK will continue to try to reconnect. If the user network recovers, the S
};
chat.on(TencentCloudChat.EVENT.NET_STATE_CHANGE, onNetStateChange);
```

Deinitialization

Destroy the SDK instance. The SDK will first log out, then disconnect the WebSocket long connection, and release resources.

```
chat.destroy();
```

Next step, login and logout.

Login and Logout Android&iOS&Windows&Mac

Last updated：2025-01-13 16:54:11

Feature Description

After initializing the SDK, you need to call the SDK login API to authenticate your account to get the permissions to use message, conversation, and other features.

Note：

1. All SDK feature APIs can be called only after successful login, except the APIs to get the conversation list and pull historical messages. Therefore, **make sure that you have logged in successfully** before using other features.
2. You can call the APIs to get the conversation list and pull historical messages even if your login failed. In this case, the locally cached conversation list and historical messages will be returned and can be displayed when there is no network connection.

Login

Your first login to an Chat account doesn't require signup, as Chat will automatically sign the account up after finding out that you are using a new account. You can call the `login` API (Java / Swift / Objective-C / C++) to log in.

Key parameters of the `login` API are as follows:

Parameter	Definition	Description
UserID	Unique user ID	It can contain up to 32 bytes of letters (a-z and A-Z), digits (0-9), underscores (_), and hyphens (-).
UserSig	Login ticket	It is calculated by your business server to ensure security. For more information, see Generating UserSig.

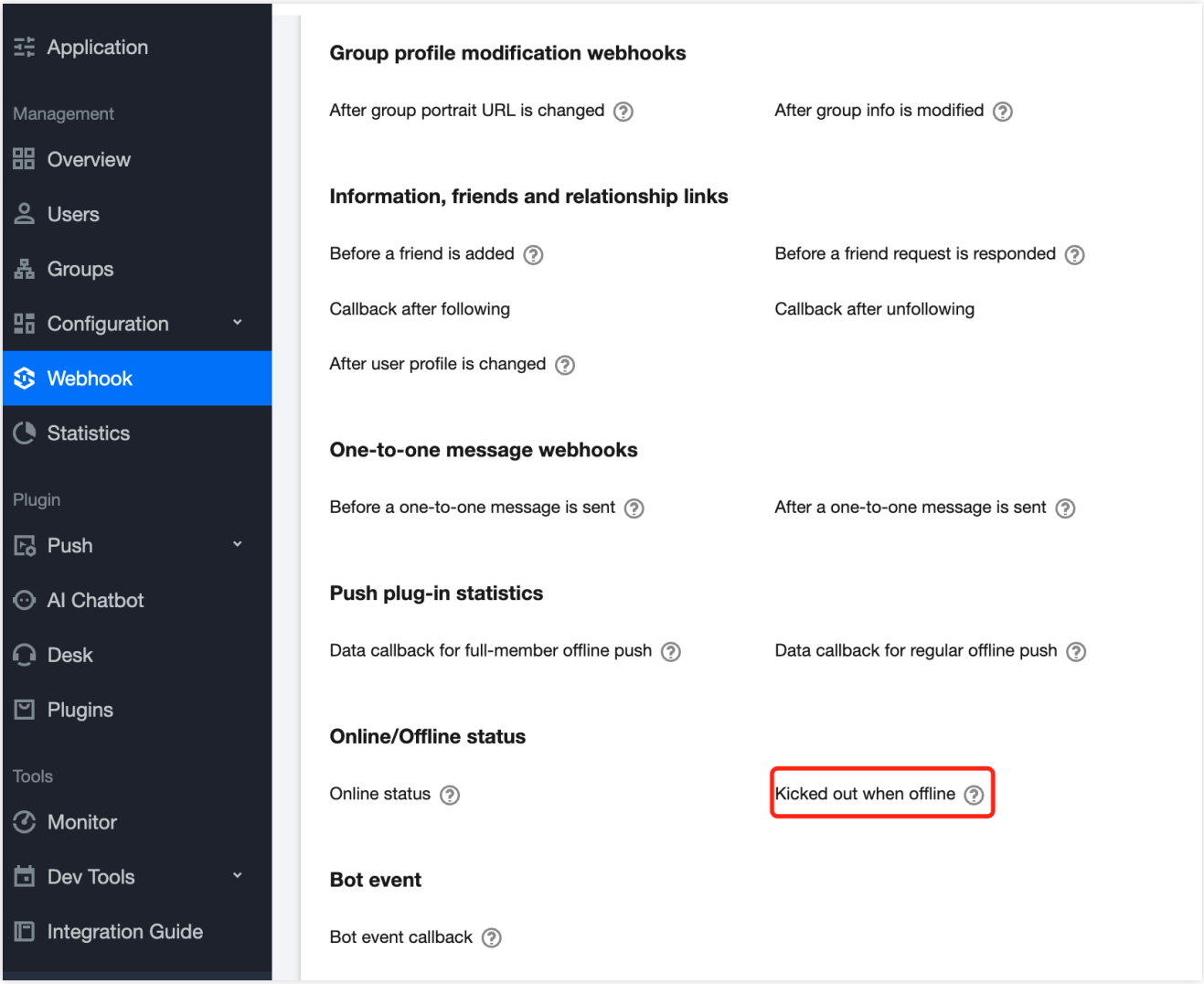
You can call the `login` API in the following scenarios:

You use the features of the SDK for the first time after your application is started.

Your ticket expires on login: The callback of the `login` API returns the `ERR_USER_SIG_EXPIRED (6206)` or `ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)` error code. In this case, you need to generate a new `userSig` to log in again.

Your ticket expires when the user is online: You may receive the `onUserSigExpired` callback (Java / Swift / Objective-C / C++) when users are online. In this case, you need to generate a new `userSig` to log in again.

A user is kicked offline: When a user is kicked offline, the SDK will notify you through the `onKickedOffline` callback (Java / Swift / Objective-C / C++). In this case, you can prompt the user and call `login` to log in again.
Kicked off while offline: If a user is kicked off while offline, they will log in successfully by default when they come back online. If you need to return the error code ERR_LOGIN_KICKED_OFF_BY_OTHER（6208）for being kicked off offline, you need to configure it separately in the console.



You don't need to call the `login` API in the following scenarios:
After the user's network is disconnected and then reconnected, you don't need to call the `login` function, as the SDK will automatically go online.
The login process is running.

Note：

1. After you call the SDK API and log in successfully, MAU calculation will start; therefore, call the login API as needed to avoid a high MAU.

©2013-2025 Tencent Cloud International Pte. Ltd.

2. You cannot log in to multiple SDK accounts of the same application at the same time; otherwise, only the last logged in account will be online.

Sample code:

Java

Swift

Objective-C

C++

```
String userID = "your user id";
String userSig = "userSig from your server";
V2TIMManager.getInstance().login(userID, userSig, new V2TIMCallback() {
    @Override
    public void onSuccess() {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        // The following error codes indicate an expired `userSig`, and you need to
        // 1. ERR_USER_SIG_EXPIRED (6206)
        // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
        // Note: Do not call the login API in case of other error codes; otherwise,
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
let userID = "your user id";
let userSig = "userSig from your server";
V2TIMManager.shared.login(userID: userID, userSig: userSig) {
    print("login succ")
} fail: { code, message in
    // The following error codes indicate an expired `userSig`, and you need to gen
    // 1. ERR_USER_SIG_EXPIRED (6206)
    // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
    // Note: Do not call the login API in case of other error codes; otherwise, the
    if code == 6206 || code == 70001 {
        print("UserSig expired, please get a new UserSig.")
    } else {
        print("Login failure, code: \\(code), message: \\(message)")
    }
}
```

```
NSString *userID = @"your user id";
NSString *userSig = @"userSig from your server";
```

```
[[V2TIMManager sharedInstance] login:userID userSig:userSig succ:^{
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    // The following error codes indicate an expired `userSig`, and you need to gen
    // 1. ERR_USER_SIG_EXPIRED (6206)
    // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
    // Note: Do not call the login API in case of other error codes; otherwise, the
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
class LoginCallback final : public V2TIMCallback {
public:
    LoginCallback() = default;
    ~LoginCallback() override = default;

    using SuccessCallback = std::function<void()>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }
    void OnSuccess() override {
        if (success_callback_) {
            success_callback_();
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMString userID = "your user id";
V2TIMString userSig = "userSig from your server";
auto callback = new LoginCallback;
callback->SetCallback(
    [=]() {
        std::cout << "success";
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
```

```
            // The following error codes indicate an expired `userSig`, and you need
        // 1. ERR_USER_SIG_EXPIRED (6206)
        // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
        // Note: Do not call the login API in case of other error codes; otherwise,
         std::cout << "failure, code:" << error_code << ", desc:" << error_message.C
         delete callback;
    });
V2TIMManager::GetInstance()->Login(userID, userSig, callback);
```

Getting the UserID

After logging in successfully, call `getLoginUser` (Java / Swift / Objective-C / C++) to get the `UserID`.

If the login fails, the `UserID` will be empty.

Sample code:

Java

Swift

Objective-C

C++

```
// Get the `UserID` of the logged-in user
String loginUserID = V2TIMManager.getInstance().getLoginUser();
```

```
// Get the `UserID` of the logged-in user
let loginUserID = V2TIMManager.shared.getLoginUser()
```

```
// Get the `UserID` of the logged-in user
NSString *loginUserID = [[V2TIMManager sharedInstance] getLoginUser];
```

```
// Get the `UserID` of the logged-in user
V2TIMString loginUserID = V2TIMManager::GetInstance()->GetLoginUser();
```

Getting the login status

Call `getLoginStatus` (Java / Swift / Objective-C / C++) to get the login status. If a user is logged in or logging in, don't call the login API frequently. The SDK supports the following login statuses:

Login Status	Description
V2TIM_STATUS_LOGINED	Logged in
V2TIM_STATUS_LOGINING	Logging in
V2TIM_STATUS_LOGOUT	Not logged in

©2013-2025 Tencent Cloud International Pte. Ltd.

Sample code:

Java

Swift

Objective-C

C++

```
int loginStatus = V2TIMManager.getInstance().getLoginStatus();
```

```
let loginStatus = V2TIMManager.shared.getLoginStatus()
```

```
V2TIMLoginStatus loginStatus = [[V2TIMManager sharedInstance] getLoginStatus];
```

```
V2TIMLoginStatus loginStatus = V2TIMManager::GetInstance()->GetLoginStatus();
```

Multi-Client Login and Kickout

You can configure multi-client login policies for the SDK in the Chat Console.There are multiple multi-client login policies, such as **A user can be concurrently online on a mobile or desktop platform and the web platform** or **A user can be concurrently online on all platforms**.

For more information on the configuration, see Feature Configuration.

You can configure the **Max Login Instances per User per Platform** for the SDK in the Chat Console, that is, the maximum number of instances on the same platform that can be concurrently online.This feature is only available for Pro edition、Pro Plus edition、Enterprise edition. The value for **Web** and **Android, iPhone, iPad, Windows, Mac** is 10 or 3 respectively. For more information on the configuration, see Feature Configuration.

When you call the `login` API to log in, if the limit specified by the multi-client login policy for your account is exceeded, a newly logged in instance will kick out an earlier one. If you have called `addIMSDKListener` (Java / Swift / Objective-C / C++) during the initialization to add the SDK listener, an earlier login instance will receive the `onKickedOffline` callback (Java / Swift / Objective-C / C++).

Logout

Generally, if your application's lifecycle is the same as the SDK's lifecycle, there is no need to log out before exiting the application.

In special cases, for example, if you use the SDK only after entering a specific UI and no longer use it after exiting the UI, you can call the `logout` API (Java / Swift / Objective-C/ C++) to log out of the SDK, after which you will no

©2013-2025 Tencent Cloud International Pte. Ltd.

longer receive new messages. Note that in this case, you also need to call `unInitSDK` (Java / Swift / Objective-C / C++) after the logout to uninitialize the SDK.

Sample code:

Java

Swift

Objective-C

C++

```
V2TIMManager.getInstance().logout(new V2TIMCallback() {
    @Override
    public void onSuccess() {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
V2TIMManager.shared.logout(succ: {
    print("logout succ.")
}, fail: { code, desc in
    print("logout fail, code: \\(code), desc: \\(desc)")
})
```

```
[[V2TIMManager sharedInstance] logout:^{
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
class Callback final : public V2TIMCallback {
public:
    using SuccessCallback = std::function<void()>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    Callback() = default;
    ~Callback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }
```

```
    void OnSuccess() override {
        if (success_callback_) {
            success_callback_();
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMString userID = "your user id";
V2TIMString userSig = "userSig from your server";
auto callback = new Callback{};
callback->SetCallback(
    [=]() {
        std::cout << "success";
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "failure, code:" << error_code << ", desc:" << error_message.C
        delete callback;
    });
V2TIMManager::GetInstance()->Logout(callback);
```

Account Switch

Call `login` to switch between accounts in the application.

For example, to switch the logged-in user from `alice` to `bob`, just log bob in. You don't need to explicitly call `logout alice`, as this operation will be handled automatically inside the SDK.

JavaScript

Last updated：2025-01-07 15:15:29

Feature Description

Users can send and receive messages only after they have logged in to the Chat SDK. To log in to the Tencent backend server, a user needs to provide information, including the `UserID` and `UserSig`. For more information on these parameters, see Login Authentication.

Login

Note:

1. After a successful login, you must wait for the SDK to be in the ready state (by listening for the event `TencentCloudChat.EVENT.SDK_READY`) before calling interfaces that require authentication, such as `sendMessage`.
2. By default, multi-instance login is not supported, meaning if this account is already logged in on another page, continuing to log in successfully on the current page may kick the other pages offline. When a user is kicked offline, the event `TencentCloudChat.EVENT.KICKED_OUT` is triggered, and the user can handle it accordingly after listening for the event.
3. To support multi-instance login (allowing logging in to the same account concurrently on multiple webpages), log in to the Chat Console, find the target SDKAppID, and click **Application > Configuration > Login and Message > Login Settings > Max Login Instances per User per Platform** to configure the number of instances. The configuration will take effect in 5 minutes.
4. For mini-programs, mini-games, uni-app, and **React Native** (even if packaged into a native app) platforms, logins are considered web instances.
5. If this interface returns an error code 60020, it means you have not purchased a package, or the package has expired, or the purchased package is still in configuration and has not yet taken effect. Please log in to the Chat purchase page to repurchase the package. After purchase, it will take effect within 5 minutes.
6. If this interface returns an error code 2801 on the mini-program platform, please check the mini-program trusted domain configuration.

API

```
chat.login(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
UserID	String	User ID.
UserSig	String	The password with which the user logs in to the Chat console. It is essentially the ciphertext generated by encrypting information such as the UserID. For the detailed generation method, see Generating UserSig.

Return value

`Promise`

Examples

```
let promise = chat.login({userID: 'your userID', userSig: 'your userSig'});
promise.then(function(imResponse) {
  console.log(imResponse.data);  // Logged in successfully
  if (imResponse.data.repeatLogin === true) {
  // This indicates that the account is already logged in.
    console.log(imResponse.data.errorInfo);
  }
}).catch(function(imError) {
  console.warn('login error:', imError);
});
```

Get login user information.

API

```
chat.getLoginUser();
```

Parameters

None

Return value

`String`

Examples

```
const userID = chat.getLoginUser();
```

Get server time

API

```
chat.getServerTime();
```

Parameters

None

Return value

`Number`

Examples

```
const serverTime = chat.getServerTime();
```

Whether SDK is ready

After the SDK is ready, developers can call the SDK's APIs to send messages and use various features of the SDK. Developers can also listen for:

`SDK_READY` , which is triggered when the SDK enters the ready state.

`SDK_NOT_READY` , which is triggered when the SDK enters the not ready state.

API

```
chat.isReady();
```

Parameters

None

Return value

`Boolean`

Examples

```
let isReady = chat.isReady();
```

Logout

This API is used to log out of the Chat SDK. It is usually called when you switch between accounts. This API clears the login status of the current account and all the data in the memory.

Note:

1. When calling this API, the instance publishes the `SDK_NOT_READY` event. In this case, the instance is automatically logged out and cannot receive or send messages.
2. Assume that the value of the Web configuration item in the Chat Console is greater than 1, and the same account has been used to log in to instances a1 and a2. After a1.logout() is executed, a1 will be automatically logged out and cannot receive or send messages, while a2 will not be affected.
3. Assume that the Online Web Instances is set to 2, and your account has been used to log in to instances a1 and a2. When you use this account to log in to instance a3, either a1 or a2 will be forcibly logged out. In most cases, the instance that first entered the login state is forcibly logged out. This is called kicked offline due to multi-account login. If a1 is forcibly logged out, a logout process is executed within a1 and the `KICKED_OUT` event is triggered. You can listen for this event and redirect to the login page when the event is triggered. At this time, a1 is forcibly logged out, whereas a2 and a3 can continue to run properly.

API

```
chat.logout();
```

Parameters

None

Return value

`Promise`

Examples

```
let promise = chat.logout();
promise.then(function(imResponse) {
  console.log(imResponse.data);// Logged out successfully
}).catch(function(imError) {
  console.warn('logout error:', imError);
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Terminating

You can terminate SDK instances, including logout, disconnecting persistent connection, and releasing all resources.

API

```
chat.destroy();
```

Parameters

None

Return value

`Promise`

Examples

```
let promise = chat.destroy();
```

Login Settings

By default, multi-account login is not supported. If you use an account that has been logged in on another page to log in on the current page, the account may be forcibly logged out on the other page, which will trigger the `KICKED_OUT` event. You can proceed accordingly after detecting the event through listening. Below is an example:

```
let onKickedOut = function (event) {
  console.log(event.data.type);
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_ACCOUNT (The user is forcibly logged ou
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_DEVICE (The user is forcibly logged out
  // TencentCloudChat.TYPES.KICKED_OUT_USERSIG_EXPIRED (The signature expired.)
  // TencentCloudChat.TYPES.KICKED_OUT_REST_API (The user is forcibly logged out by
)};
chat.on(TencentCloudChat.EVENT.KICKED_OUT, onKickedOut);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

Last updated：2025-04-16 14:28:46

Overview

After initializing the Chat SDK, you need to call the SDK login API to authenticate your account to get the permissions to use message, conversation, and other features.

Caution:

Only the conversation acquisition API can be called immediately after you call the login API, and other APIs can be called only after you have successfully logged in to the SDK. Therefore, **make sure that you have logged in successfully** before using other features; otherwise, such features may become abnormal or unavailable.

Login

You don't need to sign up on your first login, as you will automatically sign up if the login is successful.

You can call the `login` API (details) to log in.

Key parameters of the `login` API are as follows:

Parameter	Description	Remarks
UserID	Unique user ID	It can contain up to 32 bytes of letters (a-z and A-Z), digits (0-9), underscores (_), and hyphens (-).
UserSig	Login ticket.	It is calculated by your business server to ensure security. For more information on the calculation method, see Generating UserSig.

You can call the `login` API in the following scenarios:

When you use features of the Chat SDK for the first time after the app is started.

The ticket expires before login: The callback of the `login` API returns the error `ERR_USER_SIG_EXPIRED` (6206) or `ERR_SVR_ACCOUNT_USERSIG_EXPIRED` (70001), and you need to generate a new userSig and log in again.

The ticket expires when you are online: You may receive the `onUserSigExpired` callback (details) when you are online. In this case, you need to generate a new `userSig` to log in again.

A user is kicked offline: When a user is kicked offline, the Chat SDK will notify you through the `onKickedOffline` callback (details). In this case, you can prompt the user and call `login` to log in again.

You don't need to call the `login` API in the following scenarios:

When your network is disconnected and then reconnected. In this case, the SDK automatically goes online.

©2013-2025 Tencent Cloud International Pte. Ltd.

When a login process is already running. In this case, repeated login is unnecessary.

Note:

1. After you call the Chat SDK API and log in successfully, DAU calculation will start; therefore, call the login API as needed to avoid a high DAU.
2. You cannot log in to multiple Chat SDK accounts of the same application at the same time; otherwise, only the last logged in account will be online.

Sample code:

```
String userID = "your user id";
String userSig = "userSig from your server";
V2TimCallback res = await TencentImSDKPlugin.v2TIMManager.login(userID: userID, use
if(res.code == 0){
    // Logic for successful login
}else{
     // Logic for failed login
}
```

Getting the login user

After logging in successfully, call the `getLoginUser` API (details) to get the `UserID` .

If the login fails, the `UserID` will be empty.

Sample code:

```
    // After successful login,
    // call `getLoginUser` to get the `UserID`.
    V2TimValueCallback<String> getLoginUserRes =
        await TencentImSDKPlugin.v2TIMManager.getLoginUser();
    if (getLoginUserRes.code == 0) {
      // Obtained successfully
      getLoginUserRes.data; // `getLoginUserRes.data` gets the `UserID` of the logi
    }
```

Getting the login status

Call the `getLoginStatus` API (details) to get the login status. If a user is logged in or logging in, don't call the login API frequently. The Chat SDK supports the following login status:

Login Status	Description
V2TIM_STATUS_LOGINED (1)	Logged in
V2TIM_STATUS_LOGINING (2)	Logging in
V2TIM_STATUS_LOGOUT (3)	Not logged in

©2013-2025 Tencent Cloud International Pte. Ltd.

Sample code:

```
// After successful login,
// call `getLoginStatus` to get the status of the login user.
V2TimValueCallback<int> getLoginStatusRes =
    await TencentImSDKPlugin.v2TIMManager.getLoginStatus();
if (getLoginStatusRes.code == 0) {
  int? status = getLoginStatusRes.data; // `getLoginStatusRes.data` gets the us
  if (status == 1) {
    // Logged in
  } else if (status == 2) {
    // Logging in
  } else if (status == 3) {
    // Not logged in
  }
}
```

Multi-Client Login and Kickout

You can configure multi-client login policies for the Chat SDK in the Tencent Cloud console.
There are multiple multi-client login policies, such as **A user can be concurrently online on a mobile or desktop platform and the web platform** or **A user can be concurrently online on all platforms**.
For related configuration, see Login settings.
You can configure the **Max Login Instances per User per Platform** for the Chat SDK in the Tencent Cloud console, that is, the maximum number of instances on the same platform that can be concurrently online.
This feature is available only for the Pro edition 、 Pro Plus edition or Enterprise edition. A user can be concurrently online on up to ten clients on the web platform or on up to three clients on the Android, iPhone, iPad, Windows, or macOS platform (the maximum number of clients that can be concurrently online on Flutter is subject to the actual compilation result).
For related configuration, see Login settings.
When you call the `login` API (details) to log in, if the limit specified by the multi-client login policy for your account is exceeded, a newly logged in instance will kick out an earlier one.
The one kicked out will receive the `onKickedOffline` callback (details).

Logout

Generally, if your application's lifecycle is the same as the Chat SDK's lifecycle, there is no need to log out before exiting the application.
In special cases, for example, if you use the Chat SDK only after entering a specific UI and no longer use it after

©2013-2025 Tencent Cloud International Pte. Ltd.

exiting the UI, you can call the `logout` API (details) to log out of the SDK, after which you will no longer receive new messages. Note that in this case, you also need to call `unInitSDK` (details) after the logout to uninitialize the SDK.

Sample code:

```
V2TimCallback logoutRes = await TencentImSDKPlugin.v2TIMManager.logout();
if(logoutRes.code == 0){

}
```

Account Switch

Call `login` (details) to switch between accounts in the application.

For example, to switch the logged-in user from `alice` to `bob`, just log bob in. You don't need to explicitly call `logout alice`, as this operation will be handled automatically inside the Chat SDK.

Unity

Last updated：2025-03-20 16:19:57

Feature Description

After initializing the Chat SDK, you need to call the SDK login API to authenticate your account to get the permissions to use message, conversation, and other features.

Caution:

Only the conversation acquisition API can be called immediately after you call the login API, and other APIs can be called only after you have successfully logged in to the SDK. Therefore, **make sure that you have logged in successfully** before using other features; otherwise, such features may become abnormal or unavailable.

Login

You don't need to sign up on your first login, as you will automatically sign up if the login is successful.

You can call the `Login` API (Details) to log in.

Key parameters of the `Login` API are as follows:

Parameter	Description	Remarks
user_id	Unique user ID	It can contain up to 32 bytes of letters (a-z and A-Z), digits (0-9), underscores (_), and hyphens (-).
user_sig	Login ticket	It is calculated by your business server to ensure security. For more information, see Generating UserSig.
callback	Async callback	It can be NullValueCallback or ValueCallback `<String>` .

You can call the `Login` API in the following scenarios:

You use the features of the Chat SDK for the first time after your application is started.

Your ticket expires on login: The callback of the `Login` API returns the `ERR_USER_SIG_EXPIRED (6206)` or `ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)` error code. In this case, you need to generate a new `user_sig` to log in again.

Your ticket expires when the user is online: You might receive the `UserSigExpiredCallback` callback (Details) when users are online. In this case, you need to generate a new `user_sig` to log in again.

©2013-2025 Tencent Cloud International Pte. Ltd.

A user is kicked offline: When a user is kicked offline, the Chat SDK will notify you through the `KickedOfflineCallback` callback (Details). In this case, you can prompt the user and call `Login` to log in again.

You don't need to call the `Login` API in the following scenarios:

After the user's network is disconnected and then reconnected, you don't need to call the `Login` function, as the Chat SDK will automatically go online.

The login process is running.

Caution:

1. After you call the Chat SDK API and log in successfully, DAU calculation will start; therefore, call the login API as needed to avoid a high DAU.
2. You cannot log in to multiple Chat SDK accounts of the same application at the same time; otherwise, only the last logged in account will be online.

Sample code:

```
public static void Login() {
  if (userid == "" || user_sig == "")
  {
      return;
  }
  TIMResult res = TencentIMSDK.Login(userid, user_sig, (int code, string desc, stri
    // Process the login callback logic
  });
```

Getting the UserID

After logging in successfully, call `GetLoginUserID` (Details) to get the `UserID` .

If the login fails, the `UserID` will be empty.

Sample code:

```
public static void GetLoginUserID()
  {
    StringBuilder userId = new StringBuilder(128);

    TIMResult res = TencentIMSDK.GetLoginUserID(userId);

    Debug.Log(userId.ToString());
  }
```

Getting the login status

Call `GetLoginStatus` (Details) to get the login status. If a user is logged in or logging in, don't call the login API frequently. The IM SDK supports the following login statuses:

Login Status	Description
kTIMLoginStatus_Logined	Logged in
kTIMLoginStatus_Logining	Logging in
kTIMLoginStatus_Logouting	Logging out
kTIMLoginStatus_UnLogined	Not logged in

Sample code:

```
public static void GetLoginStatus()
 {
   TIMLoginStatus res = TencentIMSDK.GetLoginStatus();
 }
```

Multi-Client Login and Mutual Kickout

You can configure multi-client login policies for the Chat SDK in the Tencent Cloud console.
There are multiple multi-client login policies, such as **A user can be concurrently online on a mobile or desktop platform and the web platform** or **A user can be concurrently online on all platforms**.
For more information on the configuration, see Feature Configuration.
You can configure the **Max Login Instances per User per Platform** for the Chat SDK in the Tencent Cloud console, that is, the maximum number of instances on the same platform that can be concurrently online.
This feature is available only for the Pro edition 、 Pro Plus edition or Enterprise edition. A user can be concurrently online on up to ten clients on the web platform or on up to three clients on the Android, iPhone, iPad, Windows, or macOS platform (the maximum number of clients that can be concurrently online on Flutter is subject to the actual compilation result).
For more information on the configuration, see Feature Configuration.
When you call the `Login` API (Details) to log in, if the limit specified by the multi-client login policy for your account is exceeded, a newly logged in instance will kick out an earlier one.
The one kicked out will receive `KickedOfflineCallback` (Details).

Logout

Generally, if your application's lifecycle is the same as the Chat SDK's lifecycle, there is no need to log out before exiting the application.

©2013-2025 Tencent Cloud International Pte. Ltd.

In special cases, for example, if you use the Chat SDK only after entering a specific UI and no longer use it after exiting the UI, you can call the `Logout` API (Details) to log out of the SDK, after which you will no longer receive new messages. Note that in this case, you also need to call `Uninit` (Details) after the logout to uninitialize the SDK.

Sample code:

```
public static void Logout()
  {
    TIMResult res = TencentIMSDK.Logout((int code, string desc, string json_param,
    // Process the logout callback logic
    });
  }
```

Account Switch

Call `Login` (Details) to switch between accounts in the application.

For example, to switch the logged-in user from `Alice` to `Bob`, just call `Login bob`. You don't need to explicitly call `logout alice`, as this operation will be handled automatically inside the Chat SDK.

FAQs

1. What should I do if error code 6013 is returned along with the description "not initialized" when I call the login or another API?

React Native

Last updated：2025-01-07 15:15:29

Feature Description

Users can send and receive messages only after they have logged in to the Chat SDK. To log in to the Tencent backend server, a user needs to provide information, including the `UserID` and `UserSig`. For more information on these parameters, see Login Authentication.

Login

Note:

1. After a successful login, you must wait for the SDK to be in the ready state (by listening for the event `TencentCloudChat.EVENT.SDK_READY`) before calling interfaces that require authentication, such as `sendMessage`.
2. By default, multi-instance login is not supported, meaning if this account is already logged in on another page, continuing to log in successfully on the current page may kick the other pages offline. When a user is kicked offline, the event `TencentCloudChat.EVENT.KICKED_OUT` is triggered, and the user can handle it accordingly after listening for the event.
3. To support multi-instance login (allowing logging in to the same account concurrently on multiple webpages), log in to the Chat Console, find the target SDKAppID, and click **Application > Configuration > Login and Message > Login Settings > Max Login Instances per User per Platform** to configure the number of instances. The configuration will take effect in 5 minutes.
4. For mini-programs, mini-games, uni-app, and **React Native** (even if packaged into a native app) platforms, logins are considered web instances.
5. If this interface returns an error code 60020, it means you have not purchased a package, or the package has expired, or the purchased package is still in configuration and has not yet taken effect. Please log in to the Chat purchase page to repurchase the package. After purchase, it will take effect within 5 minutes.
6. If this interface returns an error code 2801 on the mini-program platform, please check the mini-program trusted domain configuration.

API

```
chat.login(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
UserID	String	User ID.
UserSig	String	The password with which the user logs in to the Chat console. It is essentially the ciphertext generated by encrypting information such as the UserID. For the detailed generation method, see Generating UserSig.

Return value

`Promise`

Examples

```
let promise = chat.login({userID: 'your userID', userSig: 'your userSig'});
promise.then(function(imResponse) {
  console.log(imResponse.data);  // Logged in successfully
  if (imResponse.data.repeatLogin === true) {
  // This indicates that the account is already logged in.
    console.log(imResponse.data.errorInfo);
  }
}).catch(function(imError) {
  console.warn('login error:', imError);
});
```

Get login user information.

API

```
chat.getLoginUser();
```

Parameters

None

Return value

`String`

Examples

```
const userID = chat.getLoginUser();
```

Get server time

API

```
chat.getServerTime();
```

Parameters

None

Return value

`Number`

Examples

```
const serverTime = chat.getServerTime();
```

Whether SDK is ready

After the SDK is ready, developers can call the SDK's APIs to send messages and use various features of the SDK. Developers can also listen for:

`SDK_READY` , which is triggered when the SDK enters the ready state.

`SDK_NOT_READY` , which is triggered when the SDK enters the not ready state.

API

```
chat.isReady();
```

Parameters

None

Return value

`Boolean`

Examples

```
let isReady = chat.isReady();
```

Logout

This API is used to log out of the Chat SDK. It is usually called when you switch between accounts. This API clears the login status of the current account and all the data in the memory.

Note:

1. When calling this API, the instance publishes the `SDK_NOT_READY` event. In this case, the instance is automatically logged out and cannot receive or send messages.
2. Assume that the value of the Web configuration item in the Chat Console is greater than 1, and the same account has been used to log in to instances a1 and a2. After a1.logout() is executed, a1 will be automatically logged out and cannot receive or send messages, while a2 will not be affected.
3. Assume that the Online Web Instances is set to 2, and your account has been used to log in to instances a1 and a2. When you use this account to log in to instance a3, either a1 or a2 will be forcibly logged out. In most cases, the instance that first entered the login state is forcibly logged out. This is called kicked offline due to multi-account login. If a1 is forcibly logged out, a logout process is executed within a1 and the `KICKED_OUT` event is triggered. You can listen for this event and redirect to the login page when the event is triggered. At this time, a1 is forcibly logged out, whereas a2 and a3 can continue to run properly.

API

```
chat.logout();
```

Parameters

None

Return value

`Promise`

Examples

```
let promise = chat.logout();
promise.then(function(imResponse) {
  console.log(imResponse.data);// Logged out successfully
}).catch(function(imError) {
  console.warn('logout error:', imError);
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Terminating

You can terminate SDK instances, including logout, disconnecting persistent connection, and releasing all resources.

API

```
chat.destroy();
```

Parameters

None

Return value

`Promise`

Examples

```
let promise = chat.destroy();
```

Login Settings

By default, multi-account login is not supported. If you use an account that has been logged in on another page to log in on the current page, the account may be forcibly logged out on the other page, which will trigger the `KICKED_OUT` event. You can proceed accordingly after detecting the event through listening. Below is an example:

```
let onKickedOut = function (event) {
  console.log(event.data.type);
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_ACCOUNT (The user is forcibly logged ou
  // TencentCloudChat.TYPES.KICKED_OUT_MULT_DEVICE (The user is forcibly logged out
  // TencentCloudChat.TYPES.KICKED_OUT_USERSIG_EXPIRED (The signature expired.)
  // TencentCloudChat.TYPES.KICKED_OUT_REST_API (The user is forcibly logged out by
)};
chat.on(TencentCloudChat.EVENT.KICKED_OUT, onKickedOut);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Message
Message Overview
Android&iOS&Windows&Mac

Last updated：2025-03-03 14:56:56

Message Class

In the Chat SDK, the message class is `V2TIMMessage` (Java / Swift / Objective-C / C++), which will be frequently used for message sending and receiving.
The `V2TIMMessage` class defines the following content:

Attribute	Definition	Description
msgID	Message ID	It is empty when a message is created and is returned synchronously when `sendMessage` is called.
timestamp	Message timestamp	The time a message is sent to the server, which can be used for message sorting
sender	`userID` of the message sender	It can be set by users and must be the same as the `userID` passed in during login.
nickName	Nickname of the message sender	It can be set by users and modified by calling `setSelfInfo` . For more information, see User Profile.
friendRemark	Friend remark of the message sender	It is used by the receiver. For example, if Alice sets Bob's remarks to `bob01` , when Bob sends a message to Alice, Alice will see that `friendRemark` in the message is `bob01` . It can be set by calling `setFriendInfo` . For more information, see Group Member Profile.
nameCard	Group name card of the sender	It applies only to group messages. For example, if Alice changes her group name card to `doctorA` and sends a message to the group, other group members will receive the message where the `nameCard` field is `doctorA` . The receiver can prioritize this field as the username

©2013-2025 Tencent Cloud International Pte. Ltd.

		for display. `nameCard` can be set by calling `setGroupMemberInfo`.
faceURL	Profile photo of the message sender	It can be set by users and can be used to download the photo.
groupID	Group ID	`groupID` is the group ID in a group message or `nil` in a one-to-one message.
userID	User ID	`userID` is the ID of the receiver in a one-to-one message or `nil` in a group message.
seq	Message serial number	The `seq` values of one-to-one messages are locally generated and may not be in ascending order and unique; the `seq` values of group messages are generated by the server and are in ascending order and unique in the group.
random	Random message code	It is generated in the SDK.
status	Message sending status	Currently, the following statuses are supported: sending, sent successfully, failed to send, deleted, imported locally, and recalled.
isSelf	Whether the message is sent by oneself	It can be used to filter messages.
needReadReceipt	Whether the message requires a read receipt	It is set by the sender and valid for the Pro edition、Pro Plus edition、Enterprise edition on v6.1 or later. For more information, see Read Receipt.
priority	Message priority	It applies only to group messages and can be set by calling the `sendMessage` API.
groupAtUserList	List of users mentioned in the group message	It applies only to group messages and stores the `userID`. For more information, see Group @ Message.
elemType	Message type	Currently, the following message types are supported: text, custom, image, audio, video, file, geographical location, emoji, group tip, and merged messages. For more information, see Message Types.

©2013-2025 Tencent Cloud International Pte. Ltd.

textElem	Text message storage element	It needs to be created and filled content when a text message is sent. When the message is received and `elemType` is identified as `TEXT`, the content can be parsed from `textElem`. For more information, see Sending Message / Receiving Message.
customElem	Custom message storage element	It needs to be created and filled content when a custom message is sent. When the message is received and `elemType` is identified as `CUSTOM`, the content can be parsed from `customElem`.
imageElem	Image message storage element	It needs to be created and filled content when an image message is sent. When the message is received and `elemType` is identified as `IMAGE`, the content can be parsed from `imageElem`.
soundElem	Audio message storage element	It needs to be created and filled content when an audio message is sent. When the message is received and `elemType` is identified as `SOUND`, the content can be parsed from `soundElem`.
videoElem	Video message storage element	It needs to be created and filled content when a video message is sent. When the message is received and `elemType` is identified as `VIDEO`, the content can be parsed from `videoElem`.
fileElem	File message storage element	It needs to be created and filled content when a file message is sent. When the message is received and `elemType` is identified as `FILE`, the content can be parsed from `fileElem`.
locationElem	Geographical location message storage element	It needs to be created and filled content when a geographical location message is sent. When the message is received and `elemType` is identified as `LOCATION`, the content can be parsed from `locationElem`.
faceElem	Emoji message storage element	It needs to be created and filled content when an emoji message is sent. When the message is received and `elemType` is identified as

©2013-2025 Tencent Cloud International Pte. Ltd.

		`FACE` , the content can be parsed from `faceElem` .
mergerElem	Merged message storage element	It needs to be created and filled content when a merged message is sent. When the message is received and `elemType` is identified as `MERGER` , the content can be parsed from `mergerElem` . For more information, see Forwarding Message.
groupTipsElem	Group tip message storage element	Currently, a group tip message can be sent only by the server. When the message is received by the SDK and `elemType` is identified as `GROUP_TIPS` , the content can be parsed from `groupTipsElem` .
cloudCustomData	Custom message data	It is set by the sender and can be customized. It is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.
isExcludedFromUnreadCount	Whether the message is excluded from the unread message count of the conversation	It is set by the sender. By default, messages are included in the unread message count. This field is supported by v5.3.425 or later.
isExcludedFromLastMessage	Whether the message is excluded from the `lastMsg` of the conversation	It is set by the sender. By default, messages are included in the `lastMsg` of the conversation. This field is supported by v5.4.666 or later. For more information, see Conversation List.
isExcludedFromContentModeration	Whether the message is excluded from content moderation （cloud moderation）	It is set by the sender. By default, messages are included in the content moderation. This field is supported by v7.1 or later. Only when cloud moderation feature is enabled, the settings will be effective.
hasRiskContent	Whether the message is identified as risk message	Only messages containing sound and video are supported. It will take effect after you enabling the cloud moderation feature.

©2013-2025 Tencent Cloud International Pte. Ltd.

		If you send a risky sound or video message, you will receive the `onRecvMessageModified` callback with this field being set to true after the cloud moderation. This field is supported by v7.4 or later.
disableCloudMessagePreHook	Whether to disable the cloud message hook before sending	It is set by the sender. By default, enable the cloud message hook before sending. This field is supported by v8.1 or later.
disableCloudMessagePostHook	Whether to disable the cloud message hook after sending	It is set by the sender. By default, enable the cloud message hook after sending. This field is supported by v8.1 or later.
isRead	Whether the message is read at the local end	A message is read by default if it is sent by oneself.
isPeerRead	Whether the message is read by the receiver	It applies only to one-to-one messages.
localCustomData	Custom message data	It is set by the sender. It is saved locally, will not be sent to the receiver, and will become invalid after the application is uninstalled and reinstalled.
localCustomInt	Custom message data	It is set by the sender. It is saved locally, will not be sent to the receiver, and will become invalid after the application is uninstalled and reinstalled. It can be used to mark whether an audio or video message has been played back.

Message Types

Messages can be classified by message destination into two types: one-to-one messages (also called C2C messages) and group messages.

Message Type	API Keyword	Description
One-to-one message	C2CMessage	It is also called the C2C message. When sending a one-to-one message, you must specify the `UserID` of the receiver, and only the receiver can receive this message.

Group message	GroupMessage	When sending a group message, you must specify the `groupID` of the target group, and all users in the group can receive this message.

Chat messages can also be classified by content into text messages, custom (signaling) messages, image messages, video messages, audio messages, file messages, location messages, and merged messages.

Message Type	API Keyword	Description
Text message	TextElem	Ordinary text message
Custom message	CustomElem	It is a section of binary buffer and often used to transfer custom signaling in your application.
Image message	ImageElem	When the SDK sends an original image, it automatically generates two images in different sizes. The three images are called the original image, large image, and thumbnail.
Video message	VideoElem	A video message contains a video file and a thumbnail.
Audio message	SoundElem	It supports displaying a red dot before the playback of the audio message.
File message	FileElem	A file message cannot exceed 100 MB.
Location message	LocationElem	A location message contains three fields: location description, longitude, and latitude.
Merged message	MergerElem	It is suitable for merging and forwarding chat history. Up to 300 messages can be merged.

Message Storage Strategy

Chat messages can be classified by message storage strategy into two types: online messages and non-online messages.

Online messages can be received only by online users and will not be pushed when they are offline. Non-online messages can be received by users whether they are online or not.

Online messages are delivered in real time and not stored on the server or in the SDK. Therefore, they cannot be pulled from historical messages after the device is changed or the application is uninstalled and reinstalled.

Note

1. All messages in an audio-video group are online messages.
2. Messages pushed to all users are online messages.

©2013-2025 Tencent Cloud International Pte. Ltd.

Non-online messages are stored in the SDK and on the server. By default, they are stored on the roaming server for seven days. If you want a longer storage period, you need to purchase the value-added service. For more information on the service content and billing, see Pricing.
These messages can be pulled from historical messages after the device is changed or the application is uninstalled and reinstalled.

©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript

Last updated：2025-01-14 15:47:12

Message Class

In the Chat SDK, `Message` indicates the message object that is used to describe the attributes of a message, such as type, content, and conversation ID.

Attribute	Type	Default Value	Description
ID	String	-	Message ID `${sende` which is the of native Cl
type	String	-	Message ty `Tencent` `Tencent` message `Tencent` message `Tencent` message `Tencent` `Tencent` message `Tencent` message `Tencent` message `Tencent` message `Tencent` group syste
payload	Object	-	Message c `TextPay` `ImagePa` `AudioPa` `VideoPa` `FilePay` `CustomP`

			`MergerP` `Locatio` `FacePay` `GroupTi` `GroupSy`
conversationID	String	-	Conversati
conversationType	String	-	Conversati `Tencent` to-one) con `Tencent` conversatic `Tencent` conversatic
to	String	-	`userID`
from	String	-	`userID` logged-in u
flow	String	-	Message fl `in` --- th `out` --- t
time	Number	-	Message ti
status	String	-	Message st `unSend` `success` `fail` ---
isRevoked	Boolean	false	Whether th indicates ye
priority	String	`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL`	Message p frequency li messages f `Tencent` `Tencent` `(default` `Tencent` `Tencent`
nick	String	-	Nickname c set by callir

avatar	String	-	Profile phot needs to be advance.
nameCard	String	-	Group nam group mess the messag calling the
atUserList	Array	-	This field st who are me
cloudCustomData	String	''	Custom me sent to the application
isDeleted	Boolean	false	Whether th indicates ye
isModified	Boolean	false	Whether th indicates ye
needReadReceipt	Boolean	false	Whether a use it, you edition or E
readReceiptInfo	Object	-	Message re `readCou` can be que `getMess` members w `getGrou` `unreadC` which can b `getMess` `isPeerR` sent a read message se roaming aft `sendMes` `timesta` a read rece when the m or pulls roa `sendMes`

isBroadcastMessage	Boolean	false	Whether th indicates ye
isSupportExtension	Boolean	false	whether the yes. To use Plus edition
revokerInfo	Object	-	Information
revokeReason	String	-	Reason for
hasRiskContent	Boolean	-	Whether im having secu If the conte compliant, t event after

TextPayload

Name	Type	Description
text	String	Text message content

ImagePayload

Name	Type	Description
uuid	String	Unique identifier of an image
imageFormat	Format	Image format type. JPG/JPEG = 1, GIF = 2, PNG = 3, BMP = 4, other = 255
imageInfoArray	Array.<Object>	Image information. Each element contains attributes described as follows: width - Number height - Number url - String - Image address, which can be used for rendering size - Number - Image size, in bytes type - Number - 0, Original image; 1, 198p compressed image; 2, 720p compressed image

AudioPayload

Name	Type	Description

uuid	String	Unique identifier
url	String	Audio address, which can be used for playback
size	Number	File size, in bytes
second	Number	Audio duration, in seconds

VideoPayload

Name	Type	Description
videoFormat	String	Video file format
videoSecond	Number	Video file duration, in seconds and of integer type
videoSize	Number	Video file size, in bytes
videoUrl	String	Video file address, which can be used for playback
videoUUID	String	Unique identifier of a video
snapshotWidth	Number	snapshot image width
snapshotHeight	Number	snapshot image height
snapshotUrl	String	snapshot image address, which can be used rendering(the tim-upload-plugin is required)

FilePayload

Name	Type	Description
uuid	String	Unique identifier
fileName	String	File name
fileUrl	String	File address
fileSize	Number	File size, in bytes

CustomPayload

Name	Type	Description
data	String	"data" field of a custom message

©2013-2025 Tencent Cloud International Pte. Ltd.

description	String	"description" field of a custom message
extension	String	"extension" field of a custom message

MergerPayload

Name	Type	Description
downloadKey	String	Key for downloading a combined message. If a combined message is large in size, the SDK will store it on the cloud, using this key as the unique identifier of the message.
messageList	Array.<SimplifiedMessage>	List of messages combined.
title	String	Title of the combined message, for example, "Chat History of the Talent Center in the Greater Bay Area".
abstractList	String	Digest list. You can set digest information in different formats for different message types, for example: for a text message, the digest can be in the "sender:text" format. For an image message, the digest can be in the "sender:[image]" format. For a file message, the digest can be in the "sender:[file]" format.
compatibleText	String	Compatible text. SDKs of early versions do not support combined messages, and they will send a text message with the content `compatibleText` by default.
layersOverLimit	Boolean	Whether the number of combination nesting levels exceeds the limit. true: the limit is exceeded, and the message is truncated. Default value: false.

GeoPayload

Name	Type	Description
description	String	Related description
latitude	Number	Latitude
longitude	Number	longitude

FacePayload

Name	Type	Description

©2013-2025 Tencent Cloud International Pte. Ltd.

index	Number	Emoji index, which is customized by the user
data	String	Extra data

GroupTipPayload

Name	Type	Description
groupJoinType	Number	Group joining type. Supported values are as follows: 0 - Default value, indicating that the operation is not a group joining operation. 1 - A user requests to join the group. 2 - A user is invited to the group.
operatorID	String	ID of the user who performs the operation.
operatorInfo	Object	User information for executing this operation (supported from v3.4.2 onwards).
operationType	Number	Operation type. Supported values are as follows: 1 - A member joins the group. 2 - A member leaves the group. 3 - A group member is removed from the group. 4 - A group member is granted the group admin role. 5 - The group admin role of a group member is revoked. 6 - The group profile is modified. 7 - The profile of a group member is modified. For example, a group member is muted. 10 - An audio-video group member is banned. 11 - An audio-video group member is unbanned.
userIDList	Array.<String>	List of relevant userIDs.
newGroupProfile	Object	If a group profile is modified, this field stores the new group profile.
memberList	Array.<Object>	If a group member is muted, this field stores related information. Each element contains attributes described as follows: `userID` - String - Group member userID `muteTime` - Number - Muting duration, in seconds `nick` - String - Nickname of the group member `avatar` - String - Avatar of group members `duration` - Number - Ban duration, unit: seconds `reason` - String - Reason for ban

GroupSystemNoticePayload

©2013-2025 Tencent Cloud International Pte. Ltd.

Name	Type	Description
operatorID	String	ID of the user who performs the operation.
operationType	Number	Operation type. Supported values are as follows: 1 - A user requests to join the group. 2 - The request to join the group is approved. 3 - The request to join the group is rejected. 4 - A user is removed from the group. 5 - The group is deleted. 6 - The group is created. 7 - A user is invited to the group. 8 - A user leaves the group. 9 - The admin is modified. 10 - The admin is canceled. 11 - The group is repossessed. 12 - A user receives a group joining invitation, and needs to accept or reject t invitation. 13 - A user invites others to join the group, and the invitation is accepted. 14 - A user invites others to join the group, and the invitation is rejected. 15 - Read report synchronized across multiple terminals. 20 - Group message remind type synchronized across multiple terminals. 21 - ban an audio-video group member. 22 - unban an audio-video group member. 23 - a group member invites others to join the group. 255 - A custom notification is triggered.
groupProfile	Object	Profile of the relevant group.
userDefinedField	String	User-defined field.
handleMessage	Object	Remarks on the processing. For example, if user1 enters remarks on an application to join group1 that requires approval, the admin of group1 will see field in the group system message.
messageRemindType	String	`TencentCloudChat.TYPES.MSG_REMIND_ACPT_AND_NOTE` - the SD receives a message and throws a MESSAGE_RECEIVED event to notify the access side, which then sends a notification. `TencentCloudChat.TYPES.MSG_REMIND_ACPT_NOT_NOTE` - the SD receives a message and throws a MESSAGE_RECEIVED event to notify the access side, which then does not send a notification. `TencentCloudChat.TYPES.MSG_REMIND_DISCARD` - the SDK reject message. `TencentCloudChat.TYPES.NOT_RECEIVE_OFFLINE_PUSH_EXCEPT` Receive messages online, and only receive push notifications for group @ messages when offline

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

Last updated：2025-04-16 14:40:38

Message Class

In the Tencent Cloud Chat SDK, the message class is `V2TimMessage` (Details), which will be frequently used for message sending and receiving.

The `V2TimMessage` class defines the following content:

Attribute	Description	Remarks
msgID	Message ID	It is empty when a message is created and is returned synchronously when `sendMessage` is called.
timestamp	Message timestamp	The time a message is sent to the server, which can be used for message sorting
sender	`userID` of the message sender	It can be set by users and must be the same as the `userID` passed in during login.
nickName	Nickname of the message sender	It can be set by users and modified by calling `setSelfInfo` . For more information, see User Profile.
friendRemark	Remarks of the message sender's friend	It is used by the receiver. For example, if Alice sets Bob's remarks to `bob01` , when Bob sends a message to Alice, Alice will see that `friendRemark` in the message is `bob01` . It can be set by calling `setFriendInfo` .
nameCard	Group name card of the sender	It applies only to group messages. For example, if Alice changes her group name card to `doctorA` and sends a message to the group, other group members will receive the message where the `nameCard` field is `doctorA` . The receiver can prioritize this field as the username for display. This field can be set by calling `setGroupMemberInfo` .
faceURL	Profile photo of the message sender	It can be set by users and can be used to download the photo.

©2013-2025 Tencent Cloud International Pte. Ltd.

groupID	Group ID	`groupID` is the group ID in a group message or `nil` in a one-to-one message.
userID	User ID	`userID` is the ID of the receiver in a one-to-one message or `nil` in a group message.
seq	Message serial number	The `seq` values of one-to-one messages are locally generated and may not be in ascending order and unique; the `seq` values of group messages are generated by the server and are in ascending order and unique in the group.
random	Random message code	It is generated in the SDK.
status	Message sending status	Currently, the following statuses are supported: sending, sent successfully, failed to send, deleted, imported locally, and recalled.
isSelf	Whether the message is sent by oneself	It can be used to filter messages.
needReadReceipt	Whether the message requires a read receipt	It is set by the sender and valid for the Ultimate edition on v6.1 or later. For more information, see Read Receipt.
priority	Message priority	It applies only to group messages and can be set by calling the `sendMessage` API.
groupAtUserList	List of users tagged in the group message	It applies only to group messages and stores the `userID`. For more information, see createAtSignedGroupMessage method.
elemType	Message type	Currently, the following message types are supported: text, custom, image, audio, video, file, geographical location, emoji, group tip, and merged messages. For more information, see Message Type.
textElem	Text message storage element	It needs to be created and entered when a text message is sent. When the message is received and `elemType` is identified as `TEXT`, the content can be parsed from `textElem`. For more information, see sendMessage method / onRecvNewMessage property.

©2013-2025 Tencent Cloud International Pte. Ltd.

customElem	Custom message storage element	It needs to be created and entered when a custom message is sent. When the message is received and `elemType` is identified as `CUSTOM`, the content can be parsed from `customElem`.
imageElem	Image message storage element	It needs to be created and entered when an image message is sent. When the message is received and `elemType` is identified as `IMAGE`, the content can be parsed from `imageElem`.
soundElem	Audio message storage element	It needs to be created and entered when an audio message is sent. When the message is received and `elemType` is identified as `SOUND`, the content can be parsed from `soundElem`.
videoElem	Video message storage element	It needs to be created and entered when a video message is sent. When the message is received and `elemType` is identified as `VIDEO`, the content can be parsed from `videoElem`.
fileElem	File message storage element	It needs to be created and entered when a file message is sent. When the message is received and `elemType` is identified as `FILE`, the content can be parsed from `fileElem`.
locationElem	Geographical location message storage element	It needs to be created and entered when a geographical location message is sent. When the message is received and `elemType` is identified as `LOCATION`, the content can be parsed from `locationElem`.
faceElem	Emoji message storage element	It needs to be created and entered when an emoji message is created. When the message is received and `elemType` is identified as `FACE`, the content can be parsed from `faceElem`.
mergerElem	Merged message storage element	It needs to be created and entered when a merged message is sent. When the message is received and `elemType` is identified as `MERGER`, the content can be parsed from `mergerElem`. For more information, see createForwardMessage method.
groupTipsElem	Group tip message	Currently, a group tip message can be sent only by the server. When the message is received by the SDK and `elemType` is identified as `GROUP_TIPS`, the content

	storage element	can be parsed from `groupTipsElem`. For more information, see onRecvNewMessage property.
cloudCustomData	Custom message data	It is set by the sender and can be customized. It is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.
isExcludedFromUnreadCount	Whether the message is excluded from the unread message count of the conversation	It is set by the sender. By default, messages are included in the unread message count. This field is supported by v5.3.425 or later.
isExcludedFromLastMessage	Whether the message is excluded from the `lastMsg` of the conversation	It is set by the sender. By default, messages are included in the `lastMsg` of the conversation. This field is supported by v5.4.666 or later.
offlinePushInfo	Custom offline push message	For more information, see Offline Push.
isRead	Whether the message is read at the local end	A message is read by default if it is sent by oneself.
isPeerRead	Whether the message is read by the receiver	It applies only to one-to-one messages.
localCustomData	Custom message data	It is set by the sender. It is saved locally, will not be sent to the receiver, and will become invalid after the application is uninstalled and reinstalled.
localCustomInt	Custom message data	It is set by the sender. It is saved locally, will not be sent to the receiver, and will become invalid after the application is uninstalled and reinstalled. It can be used to mark whether an audio or video message has been played back.

©2013-2025 Tencent Cloud International Pte. Ltd.

Message Type

Chat messages can be classified by message destination into two types: one-to-one messages (also called C2C messages) and group messages.

Message Type	API Keyword	Description
One-to-one message	C2CMessage	It is also called the C2C message. When sending a one-to-one message, you must specify the `UserID` of the receiver, and only the receiver can receive this message.
Group message	GroupMessage	When sending a group message, you must specify the `groupID` of the target group, and all users in the group can receive this message.

Chat messages can also be classified by content into text messages, custom (signaling) messages, image messages, video messages, audio messages, file messages, location messages, and merged messages.

Message Type	API Keyword	Description
Text message	TextElem	Ordinary text message
Custom message	CustomElem	It is a section of binary buffer and often used to transfer custom signaling in your application.
Image message	ImageElem	When the SDK sends an original image, it automatically generates two images in different sizes. The three images are called the original image, large image, and thumbnail.
Video message	VideoElem	A video message contains a video file and a thumbnail.
Audio message	SoundElem	It supports displaying a red dot before the playback of the audio message.
File message	FileElem	A file message cannot exceed 100 MB.
Location message	LocationElem	A location message contains three fields: location description, longitude, and latitude.
Merged message	MergerElem	Up to 300 messages can be merged.

©2013-2025 Tencent Cloud International Pte. Ltd.

Message Storage Policy

Chat messages can be classified by message storage policy into two types: online messages and non-online messages.

Online messages can be received only by online users and will not be pushed when they are offline. Non-online messages can be received by users whether they are online or not.

Online messages are delivered in real time and not stored on the server or in the SDK. Therefore, they cannot be pulled from historical messages after the device is changed or the application is uninstalled and reinstalled.

Note:

1. All messages in an audio-video group are online messages.
2. Messages pushed to all users are online messages.

Non-online messages are stored in the SDK and on the server. By default, they are stored on the roaming server for seven days. If you want a longer storage period, you need to purchase the value-added service. For more information on the service content and billing, see Pricing.

These messages can be pulled from historical messages after the device is changed or the application is uninstalled and reinstalled.

©2013-2025 Tencent Cloud International Pte. Ltd.

Unity

Last updated：2025-03-20 16:32:18

Message Class

In the Tencent Cloud Chat SDK, the message class is `Message` (details), which will be frequently used for message sending and receiving.

Meanings of Message Fields

Field	Description
message_elem_array	List of elements in the message
message_conv_id	Conversation ID of the message
message_conv_type	Conversation type of the message
message_sender	Message sender
message_priority	Message priority
message_client_time	Client time
message_server_time	Server time
message_is_from_self	Whether the message sender is the current user
message_platform	Platform from which the message is sent
message_is_read	Whether the message is read
message_is_online_msg	Whether the message is an online message. `false` (default): Ordinary message; `true` : Online message
message_is_peer_read	Whether the message is read by the recipient
message_need_read_receipt	Whether the message requires a read receipt (to use this feature, you need to purchase the Pro edition 、 Pro Plus edition or Enterprise edition on v6.1 or later). To use this feature for group messages, go to the Chat console to set the target group type first.

©2013-2025 Tencent Cloud International Pte. Ltd.

message_status	Current status of the message
message_target_group_member_array	Specifies the targeted recipients of the group message (targeted message). This field is not supported for group @ messages, community messages, or audio-video group messages. After this field is set, the message will not be included in the unread message count of the conversation.
message_unique_id	Unique ID of the message. `kTIMMsgMsgId` is recommended.
message_msg_id	Unique message ID.
message_rand	Random code of the message
message_has_sent_receipt	Whether a read receipt is sent (valid only for group messages)
message_group_receipt_read_count	This field is an internal field and not recommended. We recommend you call `TIMMsgGetMessageReadReceipts` to get the read receipt for a group message.
message_group_receipt_unread_count	This field is an internal field and not recommended. We recommend you call `TIMMsgGetMessageReadReceipts` to get the read receipt for a group message.
message_seq	Sequence number of the message
message_custom_int	Custom integer field, which is saved locally, will not be sent to the peer end, and will become invalid after the application is uninstalled and reinstalled
message_custom_str	Custom data field, which is saved locally, will not be sent to the peer end, and will become invalid after the application is uninstalled and reinstalled
message_cloud_custom_str	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled
message_is_excluded_from_unread_count	Whether the message is excluded from the unread count of the conversation. `NO` (default): included in the unread count of the conversation; `YES` : excluded from the unread count of the conversation
message_is_forward_message	Whether the message is a forwarded message
message_group_at_user_array	List of UserIDs of users to be mentioned (@). To @all, pass in `kImSDK_MesssageAtALL` .

©2013-2025 Tencent Cloud International Pte. Ltd.

message_sender_profile	User profile of the message sender
message_sender_group_member_info	Information of the message sender in a group. This parameter is only valid in a group conversation. Currently, only the `kTIMGroupMemberInfoIdentifier` and `kTIMGroupMemberInfoNameCard` fields can be obtained. You are advised to obtain other fields via the `TIMGroupGetMemberInfoList` API.
message_offlie_push_config	Offline push settings of the message
message_excluded_from_last_message	Whether the message is used as the last message (`lastMessage`) in the conversation. `true` : No; `false` : Yes

Message Types

Chat messages are classified by message destination into two types: one-to-one messages (also called C2C messages) and group messages.

Message Type	API Keyword	Description
One-to-one message	C2CMessage	Also called C2C message. When sending a one-to-one message, you must specify the UserID of the message recipient, and only the recipient can receive this message.
Group message	GroupMessage	When sending a group message, you must specify the groupID of the target group, and all users in this group can receive this message.

Tencent Cloud Chat messages can also be classified by content into text messages, image messages, video messages, audio messages, file messages, location messages, combined messages, and group tip messages.

Message Type	API Keyword	Description
Text message	kTIMElem_Text	Ordinary text message
Custom message	kTIMElem_Custom	It is a section of binary buffer and often used to transfer custom signaling in your application.
Image message	kTIMElem_Image	When the Chat SDK sends an original image, it automatically generates two images in different sizes. The three images are called the original

©2013-2025 Tencent Cloud International Pte. Ltd.

		image, large image, and thumbnail.
Video message	kTIMElem_Video	A video message contains a video file and a thumbnail.
Audio message	kTIMElem_Sound	It supports displaying a red dot before the playback of the audio message.
File message	kTIMElem_File	A file message cannot exceed 100 MB.
Location message	kTIMElem_Location	A location message contains three fields: location description, longitude, and latitude.
Merged message	kTIMElem_Merge	Up to 300 messages can be combined.
Group tip	kTIMElem_GroupTips	A group tip is often used to carry a system notification in a group, such as a notification indicating that a member joins or leaves the group, the group description is modified, or the profile of a group member is changed.

Message Storage Policy

Tencent Cloud Chat messages can be classified by message storage policy into two types: online messages and non-online messages.

Online messages can be received only by online users and will not be pushed when they are offline. Non-online messages can be received by users whether they are online or not.

Online messages are delivered in real time and not stored on the server or in the SDK. Therefore, they cannot be pulled from historical messages after the device is changed or the application is uninstalled and reinstalled.

Note:

1. All messages in an audio-video group are online messages.
2. Messages pushed to all users are online messages.

Non-online messages are stored in the SDK and on the server. By default, they are stored on the roaming server for seven days. If you want a longer storage period, you need to purchase the value-added service. For more information on the service content and billing, see Pricing.

These messages, for example, all types of ordinary messages, can be pulled from historical messages after the device is changed or the application is uninstalled and reinstalled.

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

Last updated：2025-01-07 15:23:29

Message Class

In the Chat SDK, `Message` indicates the message object that is used to describe the attributes of a message, such as type, content, and conversation ID.

Attribute	Type	Default Value	Description
ID	String	-	Message ID `${sende` which is the of native Cl
type	String	-	Message ty `Tencent` `Tencent` message `Tencent` message `Tencent` message `Tencent` `Tencent` message `Tencent` message `Tencent` message `Tencent` message `Tencent` group syste
payload	Object	-	Message c `TextPay` `ImagePa` `AudioPa` `VideoPa` `FilePay` `CustomP`

			`MergerP` `Locatio` `FacePay` `GroupTi` `GroupSy`
conversationID	String	-	Conversati
conversationType	String	-	Conversati `Tencent` to-one) con `Tencent` conversatic `Tencent` conversatic
to	String	-	`userID`
from	String	-	`userID` logged-in u
flow	String	-	Message fl `in` --- th `out` --- t
time	Number	-	Message ti
status	String	-	Message st `unSend` `success` `fail` ---
isRevoked	Boolean	false	Whether th indicates ye
priority	String	`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL`	Message p frequency li messages f `Tencent` `Tencent` `(default` `Tencent` `Tencent`
nick	String	-	Nickname c set by callir

avatar	String	-	Profile phot needs to be advance.
nameCard	String	-	Group nam group mess the messag calling the
atUserList	Array	-	This field st who are me
cloudCustomData	String	''	Custom me sent to the application
isDeleted	Boolean	false	Whether th indicates ye
isModified	Boolean	false	Whether th indicates ye
needReadReceipt	Boolean	false	Whether a use it, you
readReceiptInfo	Object	-	Message re `readCou` can be que `getMess` members w `getGrou` `unreadC` which can b `getMess` `isPeerR` sent a read message se roaming aft `sendMes` `timesta` a read rece when the m or pulls roa `sendMes`
isBroadcastMessage	Boolean	false	Whether th

			indicates ye
isSupportExtension	Boolean	false	whether the yes. To use package.
revokerInfo	Object	-	Information
revokeReason	String	-	Reason for
hasRiskContent	Boolean	-	Whether im having secu If the conte compliant, t event after

TextPayload

Name	Type	Description
text	String	Text message content

ImagePayload

Name	Type	Description
uuid	String	Unique identifier of an image
imageFormat	Format	Image format type. JPG/JPEG = 1, GIF = 2, PNG = 3, BMP = 4, other = 255
imageInfoArray	Array.<Object>	Image information. Each element contains attributes described as follows: width - Number height - Number url - String - Image address, which can be used for rendering size - Number - Image size, in bytes type - Number - 0, Original image; 1, 198p compressed image; 2, 720p compressed image

AudioPayload

Name	Type	Description

uuid	String	Unique identifier
url	String	Audio address, which can be used for playback
size	Number	File size, in bytes
second	Number	Audio duration, in seconds

VideoPayload

Name	Type	Description
videoFormat	String	Video file format
videoSecond	Number	Video file duration, in seconds and of integer type
videoSize	Number	Video file size, in bytes
videoUrl	String	Video file address, which can be used for playback
videoUUID	String	Unique identifier of a video
snapshotWidth	Number	snapshot image width
snapshotHeight	Number	snapshot image height
snapshotUrl	String	snapshot image address, which can be used rendering(the tim-upload-plugin is required)

FilePayload

Name	Type	Description
uuid	String	Unique identifier
fileName	String	File name
fileUrl	String	File address
fileSize	Number	File size, in bytes

CustomPayload

Name	Type	Description
data	String	"data" field of a custom message

©2013-2025 Tencent Cloud International Pte. Ltd.

description	String	"description" field of a custom message
extension	String	"extension" field of a custom message

MergerPayload

Name	Type	Description
downloadKey	String	Key for downloading a combined message. If a combined message is large in size, the SDK will store it on the cloud, using this key as the unique identifier of the message.
messageList	Array.<SimplifiedMessage>	List of messages combined.
title	String	Title of the combined message, for example, "Chat History of the Talent Center in the Greater Bay Area".
abstractList	String	Digest list. You can set digest information in different formats for different message types, for example: for a text message, the digest can be in the "sender:text" format. For an image message, the digest can be in the "sender:[image]" format. For a file message, the digest can be in the "sender:[file]" format.
compatibleText	String	Compatible text. SDKs of early versions do not support combined messages, and they will send a text message with the content `compatibleText` by default.
layersOverLimit	Boolean	Whether the number of combination nesting levels exceeds the limit. true: the limit is exceeded, and the message is truncated. Default value: false.

GeoPayload

Name	Type	Description
description	String	Related description
latitude	Number	Latitude
longitude	Number	longitude

FacePayload

Name	Type	Description

©2013-2025 Tencent Cloud International Pte. Ltd.

index	Number	Emoji index, which is customized by the user
data	String	Extra data

GroupTipPayload

Name	Type	Description
groupJoinType	Number	Group joining type. Supported values are as follows: 0 - Default value, indicating that the operation is not a group joining operation. 1 - A user requests to join the group. 2 - A user is invited to the group.
operatorID	String	ID of the user who performs the operation.
operatorInfo	Object	User information for executing this operation (supported from v3.4.2 onwards).
operationType	Number	Operation type. Supported values are as follows: 1 - A member joins the group. 2 - A member leaves the group. 3 - A group member is removed from the group. 4 - A group member is granted the group admin role. 5 - The group admin role of a group member is revoked. 6 - The group profile is modified. 7 - The profile of a group member is modified. For example, a group member is muted. 10 - An audio-video group member is banned. 11 - An audio-video group member is unbanned.
userIDList	Array.<String>	List of relevant userIDs.
newGroupProfile	Object	If a group profile is modified, this field stores the new group profile.
memberList	Array.<Object>	If a group member is muted, this field stores related information. Each element contains attributes described as follows: `userID` - String - Group member userID `muteTime` - Number - Muting duration, in seconds `nick` - String - Nickname of the group member `avatar` - String - Avatar of group members `duration` - Number - Ban duration, unit: seconds `reason` - String - Reason for ban

GroupSystemNoticePayload

Name	Type	Description
operatorID	String	ID of the user who performs the operation.
operationType	Number	Operation type. Supported values are as follows: 1 - A user requests to join the group. 2 - The request to join the group is approved. 3 - The request to join the group is rejected. 4 - A user is removed from the group. 5 - The group is deleted. 6 - The group is created. 7 - A user is invited to the group. 8 - A user leaves the group. 9 - The admin is modified. 10 - The admin is canceled. 11 - The group is repossessed. 12 - A user receives a group joining invitation, and needs to accept or reject t invitation. 13 - A user invites others to join the group, and the invitation is accepted. 14 - A user invites others to join the group, and the invitation is rejected. 15 - Read report synchronized across multiple terminals. 20 - Group message remind type synchronized across multiple terminals. 21 - ban an audio-video group member. 22 - unban an audio-video group member. 23 - a group member invites others to join the group. 255 - A custom notification is triggered.
groupProfile	Object	Profile of the relevant group.
userDefinedField	String	User-defined field.
handleMessage	Object	Remarks on the processing. For example, if user1 enters remarks on an application to join group1 that requires approval, the admin of group1 will see field in the group system message.
messageRemindType	String	`TencentCloudChat.TYPES.MSG_REMIND_ACPT_AND_NOTE` - the SD receives a message and throws a MESSAGE_RECEIVED event to notify the access side, which then sends a notification. `TencentCloudChat.TYPES.MSG_REMIND_ACPT_NOT_NOTE` - the SD receives a message and throws a MESSAGE_RECEIVED event to notify the access side, which then does not send a notification. `TencentCloudChat.TYPES.MSG_REMIND_DISCARD` - the SDK reject message. `TencentCloudChat.TYPES.NOT_RECEIVE_OFFLINE_PUSH_EXCEPT` Receive messages online, and only receive push notifications for group @ messages when offline

©2013-2025 Tencent Cloud International Pte. Ltd.

Sending Message Android&iOS&Windows&Mac

Last updated：2025-01-06 17:58:18

Basic Feature Description

The method for sending a message is in the core classes `V2TIMManager` and `V2TIMMessageManager (Java)` / `V2TIMManager+Message (Swift and Objective-C)` .
It supports sending text, custom, and rich media messages, and all of which belong to the `V2TIMMessage` type. `V2TIMMessage` can contain `V2TIMElem` sub-types to indicate different types of messages.

API Description

The `sendMessage` API (Java / Swift / Objective-C / C++) is one of the core APIs for message sending. It supports sending messages of all types.
Note
The advanced message sending API mentioned below refers to `sendMessage` .
The API is as described below:
Java
Swift
Objective-C
C++

```
// V2TIMMessageManager
public abstract String sendMessage(
    V2TIMMessage message,
    String receiver,
    String groupID,
    int priority,
    boolean onlineUserOnly,
    V2TIMOfflinePushInfo offlinePushInfo,
    V2TIMSendCallback<V2TIMMessage> callback);
```

```
public func sendMessage(message: V2TIMMessage,
                        receiver: String?,groupID: String?,
                        priority: V2TIMMessagePriority,
                        onlineUserOnly: Bool,
```

```
                        offlinePushInfo: V2TIMOfflinePushInfo?,
                        progress: V2TIMProgress?,
                        succ: V2TIMSucc?,
                        fail: V2TIMFail?) -> String?
```

```
// V2TIMManager+Message.h
- (NSString *)sendMessage:(V2TIMMessage *)message
                 receiver:(NSString *)receiver
                  groupID:(NSString *)groupID
                 priority:(V2TIMMessagePriority)priority
           onlineUserOnly:(BOOL)onlineUserOnly
          offlinePushInfo:(V2TIMOfflinePushInfo *)offlinePushInfo
                 progress:(V2TIMProgress)progress
                     succ:(V2TIMSucc)succ
                     fail:(V2TIMFail)fail;
```

```
// V2TIMMessageManager
virtual V2TIMString SendMessage(
    V2TIMMessage& message,
    const V2TIMString& receiver,
    const V2TIMString& groupID,
    V2TIMMessagePriority priority,
    bool onlineUserOnly,
    const V2TIMOfflinePushInfo& offlinePushInfo,
    V2TIMSendCallback* callback);
```

Parameter description:

Parameter	Definition	Valid for One-to-One Chat	Valid for Group Chat	Description
message	Message object	YES	YES	It needs to be created through the `createXxxMessage` API. Here, `Xxx` indicates the specific type.
receiver	`userID` of the one-to-one message receiver	YES	NO	Just specify `receiver` for sending one-to-one messages.
groupID	`groupID` of	NO	YES	Just specify `groupID` for sending group messages.

	the group chat			
priority	Message priority	NO	YES	Set a higher priority for important messages (such as red packets and gifts) and a lower priority for frequent and unimportant messages (such as likes).
onlineUserOnly	Whether the message can be received by online users only	YES	YES	If it is set to `true`, the message cannot be pulled when a receiver pulls historical messages. This is often used to implement weak prompts, such as "The other party is typing..." and unimportant prompts in a group.
offlinePushInfo	Offline push message	YES	YES	The title and content carried when a message is pushed offline.
progress	File upload progress	YES	YES	File upload progress. It applies to sending messages that contain rich media such as images, audios, videos, and files. There is no callback for pure text, emoji, and location messages.
succ	Callback for successful message sending	YES	YES	-
fail	Callback for failed message sending	YES	YES	Callback failure error code and error description.

Caution

If both `groupID` and `receiver` are set, targeted group messages are sent to `receiver`. For more information, see Targeted Group Message.

Sending a Text Message

Text messages include one-to-one messages and group messages, which are different in terms of API and parameters.

The ordinary and advanced APIs can be used to send text messages. The latter supports more sending parameters (such as priority and offline push message).

The ordinary API is as described below, while the advanced API is `sendMessage` mentioned above.

©2013-2025 Tencent Cloud International Pte. Ltd.

One-to-one text message

Basic API

Call the `sendC2CTextMessage` API (Java / Swift / Objective-C / C++) to send a one-to-one text message simply by passing in the message content and receiver's `userID` .

Sample code:

Java

Swift

Objective-C

C++

```
// `msgID` returned by the API for on-demand use
String msgID = V2TIMManager.getInstance().sendC2CTextMessage("One-to-one text messa
    @Override
    public void onSuccess(V2TIMMessage message) {
        // The one-to-one text message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the one-to-one text message
    }
});
```

```
// `msgID` returned by the API for on-demand use
let msgID = V2TIMManager.shared.sendC2CTextMessage(text: "this is c2c message", to:
        print("send c2c text message succ.")
    } fail: { code, desc in
        print("send c2c text message fail, code: \\(code), desc: \\(desc)")
    }
```

```
// `msgID` returned by the API for on-demand use
NSString *msgID = [[V2TIMManager sharedInstance] sendC2CTextMessage:@"One-to-one te
                                                                 to:@"receiver_user
                                                               succ:^{
    // The one-to-one text message sent successfully
} fail:^(int code, NSString *msg) {
    // Failed to send the one-to-one text message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
```

```
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The one-to-one text message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the one-to-one text message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the text message.
```

```
    });
// `msgID` returned by the API for on-demand use
V2TIMString msgID =
    V2TIMManager::GetInstance()->SendC2CTextMessage("One-to-one text message", "rec
```

Advanced API

The advanced API can be called to send a one-to-one text message in two steps:

1. Call `createTextMessage` (Java / Swift / Objective-C / C++) to create a text message.
2. Call `sendMessage` (Java / Swift / Objective-C / C++) to send the message.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a text message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createTextMessage("con
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "userID", null, V2TIMMes
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the text message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The text message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the text message
    }
});
```

```
// Create a text message
let msg = V2TIMManager.shared.createTextMessage(text: "content") {
// Send the message
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", groupID:
        .V2TIM_PRIORITY_DEFAULT, onlineUserOnly: false, offlinePushInfo: nil) { pro
    } succ: {
        print("createTextMessage & send succ.")
    } fail: { code, desc in
```

```
        print("createTextMessage &send c2c text message fail, code: \\(code), desc:
    }
}
```

```
// Create a text message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createTextMessage:@"content"
// Send the message
[V2TIMManager.sharedInstance sendMessage:message
                                receiver:@"userID"
                                 groupID:nil
                                priority:V2TIM_PRIORITY_NORMAL
                          onlineUserOnly:NO
                         offlinePushInfo:nil
                                progress:nil
                                succ:^{
    // The text message sent successfully
}
                                fail:^(int code, NSString *desc) {
    // Failed to send the text message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
```

```
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a text message
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The progress is not called back for the text message.
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the text message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the text message.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMAL, false,
```

Group text message

Basic API

Call `sendGroupTextMessage` (Java / Swift / Objective-C / C++) to send a group message simply by passing in the message content, `groupID` of the group chat, and message priority.

For message priorities, see the `V2TIMMessagePriority` definition.

Sample code:

Java

Swift

Objective-C

C++

```
// `msgID` returned by the API for on-demand use
String msgID = V2TIMManager.getInstance().sendGroupTextMessage("Group text message"
    @Override
    public void onSuccess(V2TIMMessage message) {
        // The group text message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the group text message
    }
});
```

```
// `msgID` returned by the API for on-demand use
let msgID =  V2TIMManager.shared.sendGroupTextMessage(text: "this is group text mes
```

```
// `msgID` returned by the API for on-demand use
NSString *msgID = [[V2TIMManager sharedInstance] sendGroupTextMessage:@"Group text
                                                                   to:@"groupID"  /
                                                             priority:V2TIM_PRIORIT
                                                                 succ:^{
    // The group text message sent successfully
} fail:^(int code, NSString *msg) {
    // Failed to send the group text message
}];
```

```
class SendGroupTextMessageCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendGroupTextMessageCallback() = default;
    ~SendGroupTextMessageCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallbackprogress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }
    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
```

```
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

auto callback = new SendGroupTextMessageCallback;
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The group text message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the group text message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the text message.
    });
// `msgID` returned by the API for on-demand use
V2TIMString msgID = V2TIMManager::GetInstance()->SendGroupTextMessage(
    "Group text message", "groupID", V2TIMMessagePriority::V2TIM_PRIORITY_NORMAL, c
```

Advanced API

The advanced API can be called to send a group text message in two steps:

1. Call `createTextMessage` (Java / Swift / Objective-C / C++) to create a text message.
2. Call `sendMessage` (Java / Swift / Objective-C / C++) to send the message.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a text message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createTextMessage("con
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, null, "receiver_groupID"
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the text message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The group text message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the group text message
    }
});
```

```
// Create a text message
let msg = V2TIMManager.shared.createTextMessage(text: "content") {
// Send the message
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: nil, groupID: "rece
        .V2TIM_PRIORITY_DEFAULT, onlineUserOnly: false, offlinePushInfo: nil) { pro
    } succ: {
        print("createTextMessage & send succ.")
    } fail: { code, desc in
        print("createTextMessage &send c2c text message fail, code: \\(code), desc:
    }
}
```

```
// Create a text message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createTextMessage:content];
// Send the message
[V2TIMManager.sharedInstance sendMessage:message
                                receiver:nil
                                 groupID:@"receiver_groupID"  // `groupID` of the g
                                priority:V2TIM_PRIORITY_NORMAL // Message priority
                          onlineUserOnly:NO   // For online users only
                         offlinePushInfo:nil  // Custom information for offline pus
                                progress:nil
                                    succ:^{
    // The text message sent successfully
}
```

```
                                fail:^(int code, NSString *desc) {
    // Failed to send the text message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a text message
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
// Send the message
auto callback = new SendCallback{};
```

```
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The group text message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the group text message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the text message.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, {}, "receiver_groupID", V2TIMMessagePriority::V2TIM_PRIORITY_NORM
```

Sending a Custom Message

Custom messages include one-to-one messages and group messages, which are different in terms of APIs and parameters. The ordinary and advanced APIs can be used to send custom messages.
The advanced API is `sendMessage` mentioned above (Java / Swift / Objective-C / C++), which supports more sending parameters (such as priority and offline push message) than the ordinary API.

Custom one-to-one message

Basic API

Call `sendC2CCustomMessage` (Java / Swift / Objective-C / C++) to send a custom one-to-one message simply by passing in the binary content and receiver's `userID` .
Sample code:
Java
Swift
Objective-C
C++

```
String msgID = V2TIMManager.getInstance().sendC2CCustomMessage("Custom one-to-one m
    @Override
    public void onSuccess(V2TIMMessage message) {
        // The custom one-to-one message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the custom one-to-one message
```

```
    }
});
```

```
let msgID = V2TIMManager.shared.sendC2CCustomMessage(text: "this is c2c message", t
      // The custom one-to-one message sent successfully
        print("send c2c text message succ.")
    } fail: { code, desc in
        // Failed to send the custom one-to-one message
        print("send c2c text message fail, code: \\(code), desc: \\(desc)")
    }
```

```
NSData *customData = [@"Custom one-to-one message"  dataUsingEncoding:NSUTF8StringE
NSString *msgID = [[V2TIMManager sharedInstance] sendC2CCustomMessage:customData
                                                                   to:@"receiver_us
                                                                 succ:^{
    // The custom one-to-one message sent successfully
}
                                                                 fail:^(int code, NS
    // Failed to send the custom one-to-one message
}];
```

```
class SendC2CCustomMessageCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendC2CCustomMessageCallback() = default;
    ~SendC2CCustomMessageCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }
    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
```

```
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The custom one-to-one message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the custom one-to-one message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the custom message.
    });
V2TIMString str = u8"Custom one-to-one message";
V2TIMBuffer customData = {reinterpret_cast<const uint8_t*>(str.CString()), str.Size
V2TIMString msgID = V2TIMManager::GetInstance()->SendC2CCustomMessage(customData, "
```

Advanced API

The advanced API can be called to send a custom one-to-one message in two steps:

1. Call `createCustomMessage` (Java / Swift / Objective-C / C++) to create a custom message.
2. Call `sendMessage` (Java / Swift / Objective-C / C++) to send the message.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a custom message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createCustomMessage("C
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
```

```
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the custom message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The custom one-to-one message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the custom one-to-one message
    }
});
```

```
if let msg = V2TIMManager.shared.createCustomMessage(data: "CustomMessage".data(usi
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "receiver_userID",
    } succ: {
        // The custom one-to-one message sent successfully
    } fail: { code, desc in
        // Failed to send the custom one-to-one message
    }
}
```

```
// Create a custom message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createCustomMessage("C
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the custom message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The custom one-to-one message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the custom one-to-one message
    }
});
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a custom message
V2TIMString str = u8"Custom one-to-one message";
V2TIMBuffer customData = {reinterpret_cast<const uint8_t*>(str.CString()), str.Size
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
```

```
        // The custom one-to-one message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the custom one-to-one message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the custom message.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

Custom group message

Basic API

Call `sendGroupCustomMessage` (Java / Swift / Objective-C / C++) to send a custom group message simply by passing in the binary content, `groupID` of the group chat, and priority.

For message priorities, see the `V2TIMMessagePriority` definition.

Sample code:

Java

Swift

Objective-C

C++

```
String msgID = V2TIMManager.getInstance().sendGroupCustomMessage("Custom group mess
    @Override
    public void onSuccess(V2TIMMessage message) {
        // The custom group message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the custom group message
    }
});
```

```
let customData = "customText".data(using: .utf8) ?? Data()
let msgID = V2TIMManager.shared.sendGroupCustomMessage(customData: customData, to:
                                                       priority: .V2TIM_PRIORITY_D
 succ: {
        // The custom group message sent successfully
}) { code, desc in
```

```
        // Failed to send the custom group message
}
```

```
NSData *customData = [@"Custom group message"  dataUsingEncoding:NSUTF8StringEncodi
NSString *msgID = [[V2TIMManager sharedInstance] sendGroupCustomMessage:customData
                                                                     to:@"receiver_
                                                               priority:V2TIM_PRIOR
                                                                   succ:^{
    // The custom group message sent successfully
} fail:^(int code, NSString *msg) {
    // Failed to send the custom group message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
```

```
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The custom group message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the custom group message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the custom message.
    });
V2TIMString str = u8"Custom group message";
V2TIMBuffer customData = {reinterpret_cast<const uint8_t*>(str.CString()), str.Size
V2TIMString msgID = V2TIMManager::GetInstance()->SendGroupCustomMessage(
    customData, "groupID", V2TIMMessagePriority::V2TIM_PRIORITY_NORMAL, callback);
```

Advanced API

The advanced API can be called to send a custom group message in two steps:

1. Call `createCustomMessage` (Java / Swift / Objective-C / C++) to create a custom message.
2. Call `sendMessage` (Java / Swift / Objective-C / C++) to send the message.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a custom message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createCustomMessage("C
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the custom message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        // The custom group message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the custom group message
    }
});
```

```
if let msg = V2TIMManager.shared.createCustomMessage(data: "customdata".data(using:
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: nil, groupID: "rece
    } succ: {
    // The custom group message sent successfully
    } fail: { code, desc in
    // Failed to send the custom group message
    }
}
```

```
V2TIMMessage *message = [[V2TIMManager sharedInstance] createCustomMessage:data];
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:nil
                                   groupID:@"receiver_groupID"  // `groupID` of the
                                  priority:V2TIM_PRIORITY_DEFAULT  // Message prior
                            onlineUserOnly:NO
                           offlinePushInfo:nil
                                  progress:nil
                                      succ:^{
    // The custom group message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the custom group message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
```

```
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a custom message
V2TIMString str = u8"Custom group message";
V2TIMBuffer customData = {reinterpret_cast<const uint8_t*>(str.CString()), str.Size
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The custom group message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the custom group message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the custom message.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, {}, "receiver_groupID", V2TIMMessagePriority::V2TIM_PRIORITY_NORM
```

Sending a Rich Media Message

A rich media message can be sent in the following steps:

1. Call `createXxxMessage` to create a rich media message object of a specified type. Here, `Xxx` indicates the specific message type.
2. Call `sendMessage` (Java / Swift / Objective-C / C++) to send the message.
3. Get the callback for message sending success or failure.

Image message

To create an image message, you need to get the local image path first.

During message sending, the image is uploaded to the server, and the upload progress is called back. The message is sent after the image is uploaded successfully.

Sample code:

Java

Swift

Objective-C

C++

```
// Create an image message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createImageMessage("/s
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // Image upload progress in the range of [0, 100]
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The image message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the image message
    }
});
```

```
if  let path = Bundle.main.path(forResource: "test", ofType: "png"),
    // Create an image message
    let msg = V2TIMManager.shared.createImageMessage(imagePath: path) {
    // Send the message
```

```
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", groupID:
    // Image upload progress in the range of [0, 100]
    } succ: {
    // The image message sent successfully
    } fail: { code, desc in
    // Failed to send the image message
    }
}
```

```
// Get the local image path
NSString *imagePath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"png"]
// Create an image message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createImageMessage:imagePath
// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
                                   groupID:nil
                                  priority:V2TIM_PRIORITY_DEFAULT
                            onlineUserOnly:NO
                           offlinePushInfo:nil
                                  progress:^(uint32_t progress) {
    // Image upload progress in the range of [0, 100]
} succ:^{
    // The image message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the image message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
```

```
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create an image message
V2TIMMessage v2TIMMessage =
    V2TIMManager::GetInstance()->GetMessageManager()->CreateImageMessage("./File/Xx
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The image message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the image message
        delete callback;
    },
    [=](uint32_t progress) {
        // Image upload progress in the range of [0, 100]
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

Audio message

To create an audio message, you need to get the local audio file path and audio duration first, the latter of which can be used for display on the receiver UI.

During message sending, the audio is uploaded to the server, and the upload progress is called back. The message is sent after the audio is uploaded successfully.

Sample code:

Java

Swift

Objective-C

C++

```
// Create an audio message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createSoundMessage("/s
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // Audio upload progress in the range of [0, 100]
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The audio message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the audio message
    }
});
```

```
// Get the local audio file path
func createSound() -> String? {
    let numberOne: Int = Int(arc4random())
    if let path = Bundle.main.path(forResource: "00", ofType: "caf"),
        let url = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDoma
        let data = try? Data(contentsOf: URL(fileURLWithPath: path)) {
        try? data.write(to: URL(fileURLWithPath: url))
        return url
    }

    return nil
}

func createAndSendSoundMessage() {
    if  let path = self?.createSound(),
        // Create an audio message
        let msg = V2TIMManager.shared.createSoundMessage(audioFilePath: path, durat
        // Send the message
        _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", group
```

```
        // Audio upload progress in the range of [0, 100]
        } succ: {
            print("createSoundMessage & send succ")
        } fail: { code, desc in
            print("createSoundMessage &send c2c text message fail, code: \\(code),
        }
    }
}
```

```
// Get the local audio file path
NSString *soundPath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"m4a"]
// Get the audio duration (which is only used for UI display)
AVURLAsset *asset = [AVURLAsset assetWithURL:[NSURL fileURLWithPath:soundPath]];
CMTime time = [asset duration];
int duration = ceil(time.value/time.timescale);
// Create an audio message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createSoundMessage:soundPath
// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
                                   groupID:nil
                                  priority:V2TIM_PRIORITY_DEFAULT
                            onlineUserOnly:NO
                           offlinePushInfo:nil
                                  progress:^(uint32_t progress) {
    // Audio upload progress in the range of [0, 100]
} succ:^{
    // The audio message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the audio message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
```

```
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create an audio message
V2TIMMessage v2TIMMessage =
    V2TIMManager::GetInstance()->GetMessageManager()->CreateSoundMessage("./File/Xx
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The audio message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the audio message
        delete callback;
    },
    [=](uint32_t progress) {
        // Audio upload progress in the range of [0, 100]
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

Video message

To create a video message, you need to get the local video file path, video duration, and video thumbnail first, the latter two of which can be used for display on the receiver UI.
During message sending, the video is uploaded to the server, and the upload progress is called back. The message is sent after the video is uploaded successfully.
Below is the sample code:
Java
Swift
Objective-C
C++

```
// Create a video message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createVideoMessage("/s
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // Video upload progress in the range of [0, 100]
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The video message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the video message
    }
});
```

```
// Get the local video file path
if  let snappath = Bundle.main.path(forResource: "testpng", ofType: nil),
    // Get the local video thumbnail path
    let path = Bundle.main.path(forResource: "test", ofType: "mp4"),
    // Get the video duration
    let msg = V2TIMManager.shared.createVideoMessage(videoFilePath: path, type: "mp
    // Send the message
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", groupID:n

    } succ: {
        print("createVideoMessage & send succ")
    } fail: { code, desc in
        print("createVideoMessage & send fail, \\(code), \\(desc)")
    }
```

```
}
```

```
// Get the local video file path
NSString *videoPath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"mp4"]
// Get the local video thumbnail path
NSString *snapShotPath = [[NSBundle mainBundle] pathForResource:@"testpng" ofType:@
// Get the video duration
AVURLAsset *asset = [AVURLAsset assetWithURL:[NSURL fileURLWithPath:path]];
CMTime time = [asset duration];
int duration = ceil(time.value/time.timescale);
// Create a video message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createVideoMessage:videoPath
                                                                     type:@"mp4"
                                                                 duration:duration
                                                             snapshotPath:snapShotP
// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
                                   groupID:nil
                                  priority:V2TIM_PRIORITY_DEFAULT
                            onlineUserOnly:NO
                           offlinePushInfo:nil
                                  progress:^(uint32_t progress) {
    // Video upload progress in the range of [0, 100]
} succ:^{
    // The video message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the video message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }
```

```
    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a video message
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
    "./File/Xxx.mp4", "mp4", 10, "./File/Xxx.jpg");
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The video message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the video message
        delete callback;
    },
    [=](uint32_t progress) {
        // Video upload progress in the range of [0, 100]
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

File message

To create a file message, you need to get the local file path first.

During message sending, the file is uploaded to the server, and the upload progress is called back. The message is

sent after the file is uploaded successfully.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a file message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createFileMessage("/sd
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // File upload progress in the range of [0, 100]
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The file message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the file message
    }
});
```

```
if  let path = Bundle.main.path(forResource: "test", ofType: "mp4") ,
    let msg = V2TIMManager.shared.createFileMessage(filePath: path, fileName: "name
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", groupID:

    } succ: {
    // The file message sent successfully
    } fail: { code, desc in
    // Failed to send the file message
    }
}
```

```
// Get the local file path
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"mp4"];
// Create a file message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createFileMessage:filePath f
// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
```

```
                                    groupID:nil
                                   priority:V2TIM_PRIORITY_DEFAULT
                             onlineUserOnly:NO
                            offlinePushInfo:nil
                                   progress:^(uint32_t progress) {
    // File upload progress in the range of [0, 100]
} succ:^{
    // The file message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the file message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
```

```
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a file message
V2TIMMessage v2TIMMessage =
    V2TIMManager::GetInstance()->GetMessageManager()->CreateFileMessage("./File/Xxx
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The file message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the file message
        delete callback;
    },
    [=](uint32_t progress) {
        // File upload progress in the range of [0, 100]
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

Location message

Latitude and longitude information is sent in a location message, which requires a map control for display.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a location message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createLocationMessage(
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the location message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The location message sent successfully
```

```
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the location message
    }
});
```

```
if  let msg = V2TIMManager.shared.createLocationMessage(desc: "LocationMessage text
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", groupID:

    } succ: {
        print("createLocationMessage & send succ")
    } fail: { code, desc in
        print("createLocationMessage & send fail, \\(code), \\(desc)")
    }
}
```

```
// Create a location message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createLocationMessage:@"Send
// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
                                   groupID:nil
                                  priority:V2TIM_PRIORITY_DEFAULT
                            onlineUserOnly:NO
                           offlinePushInfo:nil
                                  progress:nil
                                      succ:^{
    // The location message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the location message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
```

```
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a location message
V2TIMMessage v2TIMMessage =
    V2TIMManager::GetInstance()->GetMessageManager()->CreateLocationMessage("Locati
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The location message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the location message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the location message.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

Emoji message

Emoji codes are sent in an emoji message and need to be converted into icons by the receiver.

Sample code:

Java

Swift

Objective-C

C++

```
// Create an emoji message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createFaceMessage(1, "
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // The progress is not called back for the emoji message.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The emoji message sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the emoji message
    }
});
```

```
// Create an emoji message
if  let msg = V2TIMManager.shared.createFaceMessage(index: 0, data: nil) {
// Send the message
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "userID", groupID:
    } succ: {
    // The emoji message sent successfully
    } fail: { code, desc in
    // Failed to send the emoji message
    }
}
```

```
// Create an emoji message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createFaceMessage:1 data:[@"
// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
```

```
                                    groupID:nil
                                   priority:V2TIM_PRIORITY_DEFAULT
                                onlineUserOnly:NO
                               offlinePushInfo:nil
                                   progress:nil
                                       succ:^{
    // The emoji message sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the emoji message
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
```

```
    ProgressCallback progress_callback_;
};

// Create an emoji message
V2TIMString str = u8"tt00";
V2TIMBuffer data = {reinterpret_cast<const uint8_t*>(str.CString()), str.Size()};
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The emoji message sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the emoji message
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back for the emoji message.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

Sending a Message Containing Multiple Element Objects

To include multiple elements in a message, create a Message object and call the `appendElem` method (Java / Swift / Objective-C / C++) through the element members of the Message object to add another element member. `appendElem` only supports adding `V2TIMTextElem`, `V2TIMCustomElem`, `V2TIMFaceElem`, and `V2TIMLocationElem` after the original `V2TIMElem` (which can be any type).

Therefore, "image + text", "video + text", and "location + text" are supported, but "image + image" and "text + image" are not supported.

Sample code for "text message + custom message":

Java

Swift

Objective-C

C++

```
// Create a text message
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createTextMessage("tes
// Create a custom element
V2TIMCustomElem customElem = new V2TIMCustomElem();
```

```
customElem.setData("Custom message".getBytes());
// Add the custom element to `message.textElem`
v2TIMMessage.getTextElem().appendElem(customElem);
// Send the message
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // The progress is not called back.
    }

    @Override
    public void onSuccess(V2TIMMessage message) {
        // The message containing multiple elements sent successfully
    }

    @Override
    public void onError(int code, String desc) {
        // Failed to send the message containing multiple element objects
    }
});
```

```
// Create a text message
guard let message = V2TIMManager.shared.createTextMessage(text: "text") else {
    print("Failed to create text message.")
    return
}
// Create a custom element
let customElem = V2TIMCustomElem()
customElem.data = "Custom message".data(using: .utf8)
// Add the custom element to `message.textElem`
message.textElem?.appendElem(element: customElem)
// Send the message
V2TIMManager.shared.sendMessage(message: message, receiver: "userID", groupID: nil,

} succ: {
    // The message containing multiple elements sent successfully
    print("Custom message sent successfully.")
} fail: { code, desc in
    // Failed to send the message containing multiple element objects
    print("Failed to send custom message: \\(code), \\(desc)")
}
```

```
// Create a text message
V2TIMMessage *message = [[V2TIMManager sharedInstance] createTextMessage:@"text"];

// Create a custom element
```

```
V2TIMCustomElem *customElem = [[V2TIMCustomElem alloc] init];
customElem.data = [@"Custom message" dataUsingEncoding:NSUTF8StringEncoding];

// Add the custom element to `message.textElem`
[message.textElem appendElem:customElem];

// Send the message
[[V2TIMManager sharedInstance] sendMessage:message
                                  receiver:@"userID"
                                   groupID:nil
                                  priority:V2TIM_PRIORITY_DEFAULT
                            onlineUserOnly:NO
                           offlinePushInfo:nil
                                  progress:nil
                                      succ:^{
    // The message containing multiple elements sent successfully
} fail:^(int code, NSString *desc) {
    // Failed to send the message containing multiple element objects
}];
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
```

```
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a text message
V2TIMMessage v2TIMMessage = V2TIMManager::GetInstance()->GetMessageManager()->Creat
// Create a custom element
V2TIMCustomElem customElem;
V2TIMString str = u8"tt00";
customElem.data = {reinterpret_cast<const uint8_t*>(str.CString()), str.Size()};
// Add the custom element to `message.textElem`
v2TIMMessage.elemList.PushBack(&customElem);
// Send the message
auto callback = new SendCallback{};
callback->SetCallback(
    [=](const V2TIMMessage& message) {
        // The message containing multiple elements sent successfully
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // Failed to send the message containing multiple element objects
        delete callback;
    },
    [=](uint32_t progress) {
        // The progress is not called back.
    });
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    v2TIMMessage, "receiver_userID", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMA
```

API-related Limits

Feature	Limit Item	Description
One-to-one/group message	Content length	A one-to-one or group message must be no longer than 12KB.
	Sending frequency	One-to-one message: No limit for sending on the client; sending

		through the RESTful API is subject to the API frequency limit specified in the API documentation. Group message: Up to 40 messages can be sent per second per group (regardless of the group type, and this limit applies to each group separately).
	Receiving frequency	No limit for one-to-one messages or group messages.
	Size of a single file	SDKs support a maximum file size of 100 MB for any single file to be sent.

Note：

1. When the number of sent messages exceeds the limit, the backend will first deliver higher-priority messages, and messages with the same priority will be delivered randomly. However, if the number of high-priority messages sent in a second exceeds 40, high-priority messages will also be discarded.
2. A message that has been restricted by frequency control is not delivered or stored in the message history, but a success response will be returned to the sender. Before Group Message Is Sent webhook is triggered, but After Group Message Is Sent is not triggered.
3. The default rate limit for calling the RESTful API to send group messages is 200 times per second, which is a different concept from the aforementioned "limit of 40 messages per second for each group." Please distinguish it. Please refer to Use Limits for more limits.

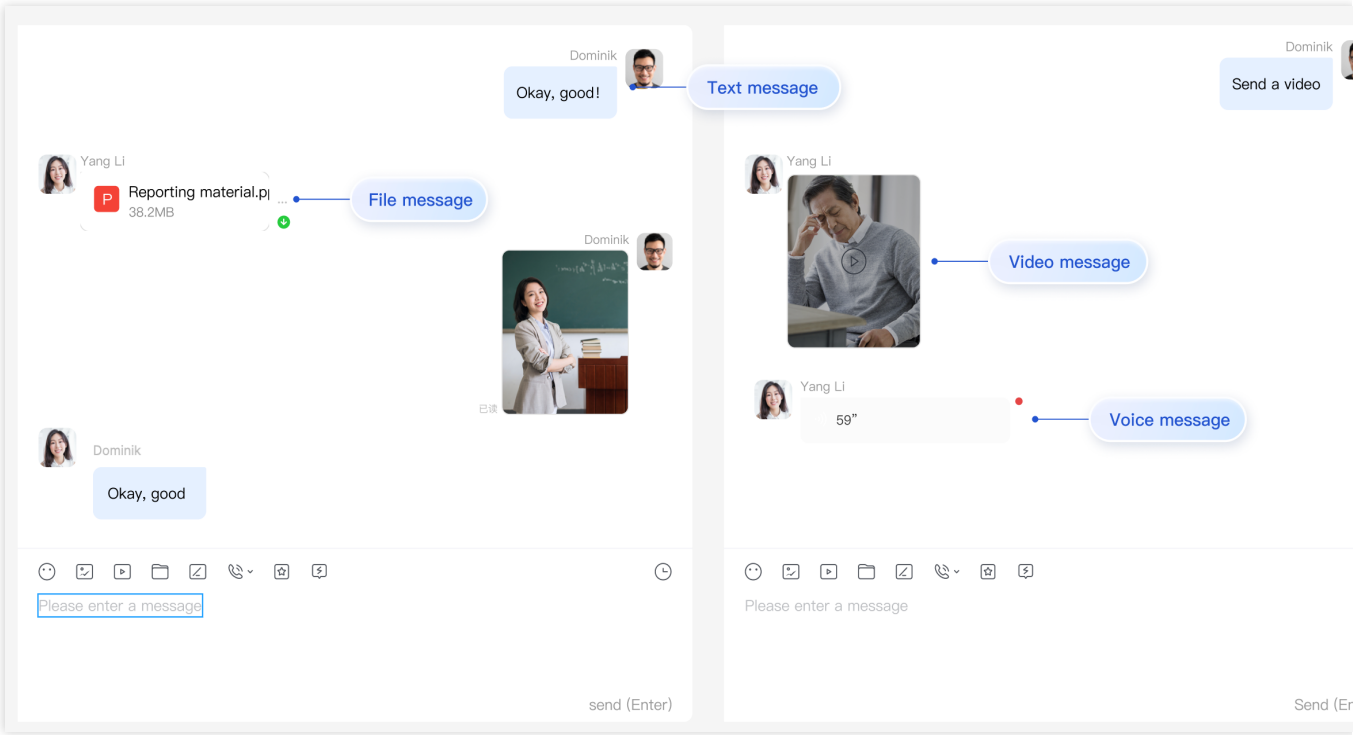
©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript

Last updated：2025-03-03 14:59:14

Chat

UI Display



Creating a Message

Creating a text message

This API is used to create a text message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a text message.

API

```
chat.createTextMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description

to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)
priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled. This attribute is supported by v2.10.2 or later
receiverList	Array | undefined	The list of group members who can receive targeted messages (not applicable to Community group and Audio-video group)
isSupportExtension	Boolean	Whether message extensions are supported: `true` (supported) or `false` (not supported) (requires you to purchase the flagship package)

The `payload` is as described below:

Name	Type	Description
text	String	Message text content

Return value

`Message`

Examples

```
// 1. Create a message instance. The returned instance can be displayed on the scre
let message = chat.createTextMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
```

©2013-2025 Tencent Cloud International Pte. Ltd.

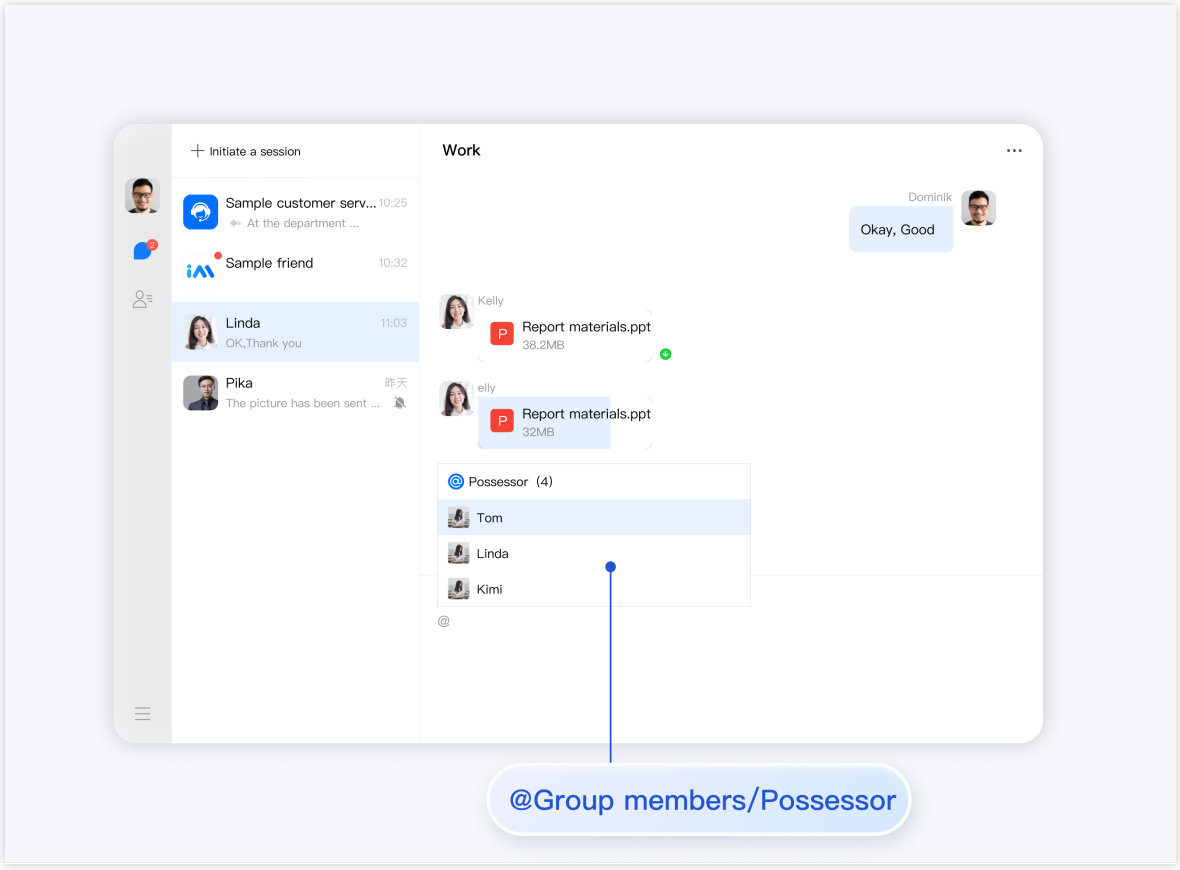
```
  payload: {
    text: 'Hello world!'
  },
  // To use the read receipt feature, you need to purchase the Pro edition 、Pro Plu
  // and set `needReadReceipt` to `true` when creating a message.
  needReadReceipt: true
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

```
// Send targeted group messages
// targeted group messages are not counted towards conversation unreadCount
// and the receiverList supports up to 50 recipients.
let message = chat.createTextMessage({
  to: 'group1',
  conversationType: TencentCloudChat.TYPES.CONV_GROUP,
  payload: {
    text: 'Hello world!'
  },
  receiverList: ['user0', 'user1']
});

let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  console.log(imResponse);
}).catch(function(imError) {
  console.warn('sendMessage error:', imError);
});
```

Creating an @mention message

This API is used to create an @mention text message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a text message.



Note:

1. It applies only to group chats, and @all members is not supported for a community and its topic.
2. Creating a group @ message does not support specifying `receiverList`.

API

```
chat.createTextAtMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)
priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`

©2013-2025 Tencent Cloud International Pte. Ltd.

		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
text	String	Text content
atUserList	Array	List of users that need to be mentioned (@). To mention (@) all, pass in `TencentCloudChat.TYPES.MSG_AT_ALL`. For example, to mention (@) `denny` and `lucy` as well as all members, pass in `['denny', 'lucy', TencentCloudChat.TYPES.MSG_AT_ALL]` for `atUserList`.

Return value

`Message`

Examples

```
// 1. Create a message instance. The returned instance can be displayed on the scre
let message = chat.createTextAtMessage({
  to: 'group1',
  conversationType: TencentCloudChat.TYPES.CONV_GROUP,
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    text: '@denny @lucy @all Dinner tonight. Reply 1 if you receive this message',
    // 'denny' and 'lucy' are `userID` values, not nicknames
    atUserList: ['denny', 'lucy', TencentCloudChat.TYPES.MSG_AT_ALL]
  },
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
```

```
  console.warn('sendMessage error:', imError);
});
```

Creating an image message

This API is used to create an image message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send an image message.

API

```
chat.createImageMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	Message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C`, `TencentCloudChat.TYPES.CONV_GROUP`.
priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
onProgress	function	Callback function for getting the upload progress
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
file	HTMLInputElement |	It is used to select a DOM node (web) or `File` object (web) of

	Object	the image. The SDK reads the data contained in this parameter and uploads the image.

Return value

`Message`

Examples

```
// Example 1: Sending an image message on web - passing in a DOM node
// 1. Create a message instance. The returned instance can be displayed on the scre
let message = chat.createImageMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    file: document.getElementById('imagePicker'),
  },
  // cloudCustomData: 'your cloud custom data'
  onProgress: function(event) { console.log('file uploading:', event) }
});

// 2. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

```
// Example 2: Sending an image message on web - passing in a File object
// Add a message input box with the ID set to `testPasteInput`
document.getElementById('testPasteInput').addEventListener('paste', function(e) {
  let clipboardData = e.clipboardData;
  let file;
  let fileCopy;
  if (clipboardData && clipboardData.files && clipboardData.files.length > 0) {
    file = clipboardData.files[0];
    // After the image message is sent successfully
    // the content pointed to by `file` may be cleared by the browser.
    // If you have extra rendering needs, you can copy the data in advance.
    fileCopy = file.slice();
  }
```

```
  if (typeof file === 'undefined') {
    console.warn('The `file` is `undefined`. Check for the compatibility of the cod
    return;
  }

  // 1. Create a message instance. The returned instance can be displayed on the sc
  let message = chat.createImageMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: {
      file: file
    },
    onProgress: function(event) { console.log('file uploading:', event) }
  });

  // 2. Send the message.
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message sent successfully
    console.log(imResponse);
  }).catch(function(imError) {
    // Failed to send the message
    console.warn('sendMessage error:', imError);
  });
});
```

```
// Sending an image on wx mini-program
wx.chooseImage({
  sourceType: ['album'],
  count: 1,
  success: function (res) {
    let message = chat.createImageMessage({
      to: 'user1',
      conversationType: TencentCloudChat.TYPES.CONV_C2C,
      payload: { file: res },
      onProgress: function(event) { console.log('file uploading:', event) }
    });

    let promise = chat.sendMessage(message);
    promise.then(function(imResponse) {
      console.log(imResponse);
    }).catch(function(imError) {
      console.warn('sendMessage error:', imError);
    });
  }
})
```

```
// Sending an image on uni-app
uni.chooseMedia({
  count: 1,
  mediaType: ['image'],
  sizeType: ['original', 'compressed'],
  sourceType: ['album'],
  success: function(res) {
    let message = chat.createImageMessage({
      to: 'user1',
      conversationType: TencentCloudChat.TYPES.CONV_C2C,
      payload: { file: res },
      onProgress: function(event) { console.log('file uploading:', event) }
    });

    let promise = chat.sendMessage(message);
    promise.then(function(imResponse) {
      console.log(imResponse);
    }).catch(function(imError) {
      console.warn('sendMessage error:', imError);
    });
  }
});
```

Creating an audio message

This API is used to create an audio message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send an audio message.

API

```
chat.createAudioMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	Message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated

		values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
file	Object	Audio file obtained after recording

Return value

`Message`

Examples

```
// Sending an audio on wx mini-program
const recorderManager = wx.getRecorderManager();

const recordOptions = {
  duration: 60000,
  sampleRate: 44100,
  numberOfChannels: 1,
  encodeBitRate: 192000,
  format: 'aac'
};


recorderManager.onError(function(errMsg) {
  console.warn('recorder error:', errMsg);
});
recorderManager.onStop(function(res) {
  console.log('recorder stop', res);

  const message = chat.createAudioMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
```

```
    payload: {
      file: res
    },
  });

  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    console.log(imResponse);
  }).catch(function(imError) {
    console.warn('sendMessage error:', imError);
  });
});

recorderManager.start(recordOptions);
```

```
// Sending an audio on web
let recorder = new Recorder({
  sampleBits: 16,
  sampleRate: 16000,
  numChannels: 1,
});
let startTs;
recorder.start().then(() => {
  startTs = Date.now();
}, (error) => {
  console.log(`${error.name} : ${error.message}`);
});

recorder.stop();

let duration = Date.now() - startTs; // unit：ms
let wavBlob = recorder.getWAVBlob();

// blob -> File
let audioFile = new File([wavBlob], 'hello.wav', { type: 'wav' });
audioFile.duration = duration;

let message = chat.createAudioMessage({
  to: 'user1',
  conversationType: 'C2C',
  payload: {
    file: audioFile
  }
});

let promise = chat.sendMessage(message);
```

```
promise.then(function(imResponse) {
  console.log(imResponse);
}).catch(function(imError) {
  console.warn('sendMessage error:', imError);
});
```

Creating a video message

This API is used to create a video message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a video message.

API

```
chat.createVideoMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	Message receiver

conversationType	String	Conversation type. Valid values:
		`TencentCloudChat.TYPES.CONV_C2C`
		`TencentCloudChat.TYPES.CONV_GROUP`

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description

file	HTMLInputElement | File | Object	Video file obtained after recording.

Return value

`Message`

Examples

```
// Sending a video message on wx mini-program
wx.chooseVideo({
  sourceType: ['album', 'camera'],
  maxDuration: 60,
  camera: 'back',
  success (res) {
    let message = chat.createVideoMessage({
      to: 'user1',
      conversationType: TencentCloudChat.TYPES.CONV_C2C,
      payload: {
        file: res
      },
      // cloudCustomData: 'your cloud custom data'
      onProgress: function(event) { console.log('file uploading:', event) }
    })

    let promise = chat.sendMessage(message);
    promise.then(function(imResponse) {
      console.log(imResponse);
    }).catch(function(imError) {
      console.warn('sendMessage error:', imError);
    });
  }
})
```

```
// Example: Sending a video message on web
// 1. Get the video file by passing in the DOM node.
// 2. Create a message instance.
const message = chat.createVideoMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    file: document.getElementById('videoPicker') // Alternatively, use `event.targe
  },
  onProgress: function(event) { console.log('file uploading:', event) }
});
// 3. Send the message.
```

```
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

```
// Sending a video message on uni-app
uni.chooseVideo({
  count: 1,
  sourceType: ['camera', 'album'],
  maxDuration: 60,
  camera: 'back',
  success: function(res) {
    let message = chat.createVideoMessage({
      to: 'user1',
      conversationType: TencentCloudChat.TYPES.CONV_C2C,
      payload: { file: res },
      onProgress: function(event) { console.log('file uploading:', event) }
    });

    let promise = chat.sendMessage(message);
    promise.then(function(imResponse) {
      console.log(imResponse);
    }).catch(function(imError) {
      console.warn('sendMessage error:', imError);
    });
  }
})
```

Creating a custom message

This API is used to create a custom message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a custom message. When the current SDK capabilities cannot meet your needs, you can customize messages.

API

```
chat.createCustomMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
data	String	Data of the custom message
description	String	Description of the custom message
extension	String	Extension of the custom message

Return value

`Message`

Examples

```
// Example: Using a custom message to implement dice rolling
// 1. Define the random function.
function random(min, max) {
  return Math.floor(Math.random() * (max - min + 1) + min);
}
// 2. Create a message instance. The returned instance can be displayed on the scre
let message = chat.createCustomMessage({
  to: 'user1',
```

```
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    data: 'dice', // Used to identify the message as a dice message
    description: String(random(1,6)), // Get the outcome
    extension: ''
  }
});
// 3. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Creating an emoji message

This API is used to create an emoji message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send an emoji message.

API

```
chat.createFaceMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

©2013-2025 Tencent Cloud International Pte. Ltd.

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
index	Number	Emoji index, which is customized by the user
data	String	Extra data

Return value

`Message`

Examples

```
// Send an emoji message on web
// 1. Create a message instance. The returned instance can be displayed on the scre
let message = chat.createFaceMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    index: 1, // Number. Emoji index, which is customized by the user
    data: 'tt00' // String. Extra data
  },
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Creating a file message

This API is used to create a file message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a file message.

API

```
chat.createFileMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (client to client conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
onProgress	function	Callback function for getting the upload progress
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
file	HTMLInputElement | File | Object	It is used to select a DOM node or `File` object of the file on web, or the success callback parameter of the uni.chooseFile API. The SDK reads the data contained in this parameter and uploads the file.

Return value

`Message`

Examples

```
// Example 1: Sending a file message on web - passing in a DOM node
// 1. Create a file message instance. The returned instance can be displayed on the
let message = chat.createFileMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    file: document.getElementById('filePicker'),
  },
  // cloudCustomData: 'your cloud custom data'
  onProgress: function(event) { console.log('file uploading:', event) }
});

// 2. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

```
// Example 2: Sending a file message on web - passing in a File object
// Add a message input box with the ID set to `testPasteInput`,
  let clipboardData = e.clipboardData;
  let file;
  let fileCopy;
  if (clipboardData && clipboardData.files && clipboardData.files.length > 0) {
    file = clipboardData.files[0];
    // After the image message is sent successfully
    // the content pointed to by `file` may be cleared by the browser.
    // If you have extra rendering needs, you can copy the data in advance.
    fileCopy = file.slice();
  }

  if (typeof file === 'undefined') {
    console.warn('The `file` is `undefined`. Check for the compatibility of the cod
    return;
  }

  // 1. Create a message instance. The returned instance can be displayed on the sc
  let message = chat.createFileMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
```

```
    payload: {
      file: file
    },
    onProgress: function(event) { console.log('file uploading:', event) }
  });

  // 2. Send the message.
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // The message sent successfully
    console.log(imResponse);
  }).catch(function(imError) {
    // Failed to send the message
    console.warn('sendMessage error:', imError);
  });
});
```

```
// Sending a file message on uni-app
uni.chooseFile({
  count: 1,
  extension:['.zip','.doc'],
  success: function(res) {
    let message = chat.createFileMessage({
      to: 'user1',
      conversationType: TencentCloudChat.TYPES.CONV_C2C,
      payload: { file: res },
      onProgress: function(event) { console.log('file uploading:', event) }
    });

    let promise = chat.sendMessage(message);
    promise.then(function(imResponse) {
      console.log(imResponse);
    }).catch(function(imError) {
      console.warn('sendMessage error:', imError);
    });
  }
});
```

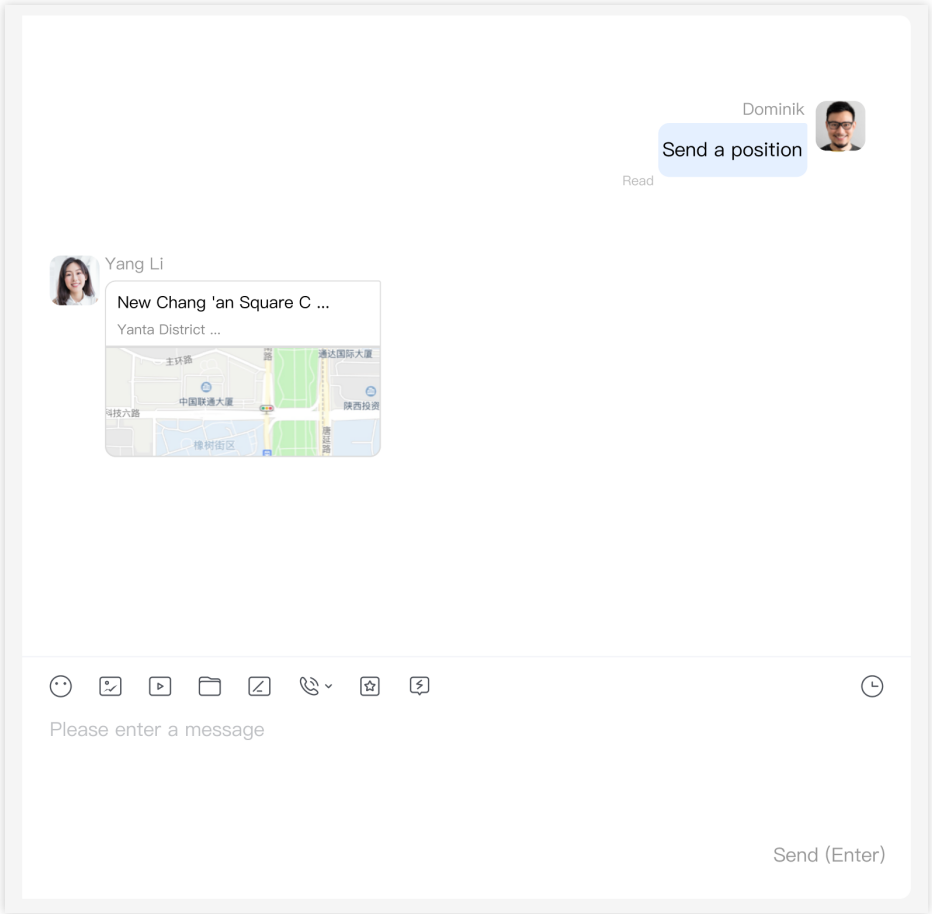
```
// Sending a file message on mini-program
wx.chooseMessageFile({
  count: 1,
  type: 'all',
  success: (res) => {
    const message = chat.createFileMessage({
      to: 'user1',
      conversationType: TencentCloudChat.TYPES.CONV_C2C,
```

```
        payload: { file: res },
        onProgress: function(event) { console.log('file uploading:', event) }
      });

      let promise = chat.sendMessage(message);
      promise.then(function(imResponse) {
        console.log(imResponse);
      }).catch(function(imError) {
        console.warn('sendMessage error:', imError);
      });
    }
  });
```

Creating a geographical location message

This API is used to create a geographical location message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a geographical location message.



API

```
chat.createLocationMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
description	String	Description of the geographical location
longitude	Number	Longitude
latitude	Number	Latitude

Return value

`Message`

Examples

```
// Send a geographical location message on web
// 1. Create a message instance. The returned instance can be displayed on the scre
let message = chat.createLocationMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
```

```
    description: 'Tencent Building, No. 10000, Shennan Boulevard, Shenzhen',
    longitude: 113.941079, // Longitude
    latitude: 22.546103 // Latitude
  }
});
// 2. Send the message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Creating a merged message

This API is used to create a merged message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a merged message.

Note:

This API does not support merging messages that failed to be sent. If the message list contains a message that failed to be sent, the API will report an error.

API

```
chat.createMergerMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`

©2013-2025 Tencent Cloud International Pte. Ltd.

		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
messageList	Array	Merged message list
title	String	Title of merged messages, for example, "Chat History of the Talent Center in the Greater Bay Area"
abstractList	String	Abstract list. You can set abstract information in different formats for different message types, for example, in the `sender:text` format for a text message, in the `sender:[image]` format for an image message, or in the `sender:[file]` format for a file message.
compatibleText	String	Compatibility text. If the SDK on an early version does not support the merged message, the user will receive a text message with the content `${compatibleText}` by default. This field is required.

Return value

`Message`

Examples

```
// 1. Forward group messages to a one-to-one conversation.
// `message1`, `message2`, and `message3` are group messages.
let mergerMessage = chat.createMergerMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    messageList: [message1, message2, message3],
    title: 'Chat History of the Talent Center in the Greater Bay Area',
    abstractList: ['allen: 666', 'iris: [Image]', 'linda: [File]'],
    compatibleText: 'Upgrade your Chat SDK to v2.10.1 or later to view this message
  },
  // cloudCustomData: 'your cloud custom data'
});

// 2. Send the message.
```

```
let promise = chat.sendMessage(mergerMessage);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Downloading a Merged Message

This API is used to download a merged message. When the merged message sent by the sender is large in size, the SDK will store it in the cloud, and the message receiver needs to download it from the cloud before viewing it.

API

```
chat.downloadMergerMessage(message);
```

Parameters

Name	Type	Description
message	Message	Message instance

Return value

`Promise`

Examples

```
// If `downloadKey` exists, the received merged message is stored in the cloud
// and needs to be downloaded first.
if (message.type === TencentCloudChat.TYPES.MSG_MERGER && message.payload.downloadK
  let promise = chat.downloadMergerMessage(message);
  promise.then(function(imResponse) {
    // After the download is successful
    // the SDK will update information such as `message.payload.messageList`.
    console.log(imResponse.data);
  }).catch(function(imError) {
    // Download failed
    console.warn('downloadMergerMessage error:', imError);
  });
}
```

Forwarding Messages One by One

©2013-2025 Tencent Cloud International Pte. Ltd.

To forward a single message, create a message identical to the original message through the `createForwardMessage` API first, and then call the `sendMessage` API to send the message.

Note:

1. Supports single forwarding and individual forwarding.

API

```
chat.createForwardMessage(message);
```

Parameters

Name	Type	Description
message	Message	Message instance

Examples

```
let forwardMessage = chat.createForwardMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: message, // The received or sent message instance
});
// 2. Send the message.
let promise = chat.sendMessage(forwardMessage);
promise.then(function(imResponse) {
  // The message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Sending a message

This API is used to send a message. You need to call any of the following message instance creation APIs to create a message instance first before calling this API to send the message instance.

`createTextMessage`

`createTextAtMessage`

`createImageMessage`

`createAudioMessage`

`createVideoMessage`

`createCustomMessage`
`createFaceMessage`
`createFileMessage`
`createLocationMessage`
`createMergerMessage`
`createForwardMessage`

Note:

1. When this API is called to send a message instance, the SDK must be in the `ready` status; otherwise, the message instance cannot be sent. The SDK status can be obtained by listening for the `TencentCloudChat.EVENT.SDK_READY` event, which will be triggered when the SDK is in the `ready` status.
2. The `TencentCloudChat.EVENT.MESSAGE_RECEIVED` event must be listened for in order to receive newly pushed one-to-one messages, group messages, group tips, or group system notifications.
3. Messages sent by this API will not trigger the `TencentCloudChat.EVENT.MESSAGE_RECEIVED` event. Messages sent by the same account from another client (or through the RESTful API) will trigger the `TencentCloudChat.EVENT.MESSAGE_RECEIVED` event.
4. Offline push applies only to Android or iOS devices but is not supported for web.
5. `onlineUserOnly` and `messageControlInfo` cannot be used together.
6. Supports counting Community and Topic messages as read.
7. When sending a message to the [System Conversation], the SDK will return error code 2106.
8. The message body size limit is 12KB and is not adjustable; exceeding this limit will result in a message sending failure with error code 80002.
9. After enabling friend relationship checking (Chat Console > Application Configuration > Feature Configuration > Login and Messages > Friend Relationship Checking), Chat will check the friend relationship when initiating a private chat, allowing only friends to send private messages. If a stranger sends a private message, the SDK will return error code 20009.

API

```
chat.sendMessage(message, options);
```

Parameters

Name	Type	Description
message	Message	Message instance
options	Object	Message sending option (message content container), which is optional

The `options` is as described below:

Name	Type	Description
onlineUserOnly	Boolean	It specifies whether the message is sent only to online users and defaults to `false`. If it is set to `true`, the message will not be stored on the roaming server, nor counted as an unread message, nor pushed to the receiver offline. It is suitable for sending unimportant tips such as broadcast notifications. This parameter is not supported when a message is sent in an audio-video group (AVChatRoom).
offlinePushInfo	Object	For more information, see Offline Push.
messageControlInfo	Object	It is a message control configuration item.

The `offlinePushInfo` is as described below:

Name	Type	Description
disablePush	Boolean	`true`: offline push is disabled; `false` (default): offline push is enabled.
disableVoipPush	Boolean	`true` to disable VoIP push (default); `false` to enable VoIP push (enabling VoIP push requires enabling offline push simultaneously)
title	String	Offline push title. This parameter is used by both iOS and Android.
description	String	Offline push content. This parameter will overwrite the offline push display text of the message instance. If the sent message is a custom message, this parameter will overwrite `message.payload.description`. If both `description` and `message.payload.description` are left empty, the receiver cannot receive the offline push notification of the custom message.
extension	String	Content passed through by offline push.
androidInfo	Object	Android Push Configuration.
apnsInfo	Object	iOS Push Configuration.

The `androidInfo` is as described below:

Name	Type	Description

sound	String	Android offline push sound settings. Only supports Huawei, Xiaomi, and Google. For Xiaomi phones running Android 8.0 and above, you must set androidInfo.XiaoMiChannelID. For Google phones with FCM push on Android 8.0 and above systems to set sound prompts, you must set androidInfo.FCMChannelID. Custom ringtones require setting the path to the sound file. For example: androidInfo.sound = "shake.xxx" Corresponding to the uniapp "nativeResources/android/res/raw/shake.xxx" audio file. Corresponding to the native app's "/res/raw/shake.xxx" audio file.
XiaoMiChannelID	String	Notification category (Channel) adaptation field for Xiaomi phones with MIUI 10 and above.
OPPOChannelID	String	NotificationChannel adaptation field for OPPO phones running Android 8.0 and above.
FCMChannelID	String	Notification channel field for Google phones running Android 8.0 and above.
VIVOClassification	Number	VIVO phone push message classification, 0: operational message, 1: system message, default is 1.
VIVOCategory	String	Used to identify the message type on VIVO phones.
HuaWeiCategory	String	Used to identify the message type on Huawei phones.
HuaWeiImage	String	Huawei push notification bar notification image URL. The protocol used must be HTTPS, an example value: https://example.com/image.png. The image file must be less than 512KB, with a recommended size of 40dp x 40dp and rounded corners of 8dp. Images exceeding the recommended specifications may experience compression or incomplete display. The recommended image formats are JPG/JPEG/PNG.
HonorImage	String	Honor push notification bar notification image URL. The protocol used must be HTTPS, an example value: https://example.com/image.png. The icon file size must be less than 100KB, with a recommended specification size of 160px x 160px and

©2013-2025 Tencent Cloud International Pte. Ltd.

		rounded corners of 32px. Icons exceeding the recommended specifications may experience compression or incomplete display.
GoogleImage	String	Google push notification bar notification image URL. The protocol used must be HTTPS, an example value: https://example.com/image.png. The icon file size must be less than 1 MB, icons exceeding the specification size may experience compression or incomplete display.

The `apnsInfo` is as described below:

Name	Type	Description
sound	String	iOS offline push sound settings. When `apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_NO_SOUND`, it means that no sound will be played upon receipt. When `apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_DEFAULT_SOUND`, it means that the system sound will be played upon receipt. Custom ringtones require setting the path to the sound file. For example: apnsInfo.sound = "shake.xxx" Corresponding to the uniapp "nativeResources/ios/Resources/shake.xxx" file, the uniapp must be a formal package. Corresponding to the "shake.xxx" audio file in the native Xcode project.
ignoreIOSBadge	Boolean	Ignore badge count for offline push (only effective for iOS), default is `false`. When set to `true`, on the iOS receiving end, this message will not increase the unread count on the APP's application icon.
disableVoipPush	Boolean	`true` to disable VoIP push (default); `false` to enable VoIP push (enabling VoIP push requires enabling offline push simultaneously)
image	String	Identifies the APNs notification bar notification image address. When the client obtains this field, it can display the image on the pop-up window by downloading the image resource. The protocol used must be HTTPS, an example value: https://example.com/image.png Supports JPEG, GIF, PNG, with a size not exceeding 10 MB.The mutable-content attribute needs to be enabled in the Chat console to support iOS 10 Service Extension features.The resource's key value is "image".

©2013-2025 Tencent Cloud International Pte. Ltd.

The `messageControlInfo` is as described below:

Name	Type	Description
excludedFromUnreadCount	Boolean	`true` : unreadCount of the conversation is not updated (the message is stored on the roaming server); `false` (default)
excludedFromLastMessage	Boolean	`true` : lastMessage of the conversation is not updated (the message is stored on the roaming server); false (default)
excludedFromContentModeration	Boolean	`true` : the message is excluded from content moderation.

Returned Value

`Promise`

Examples

```
// If the receiver is offline, the message will be stored on the roaming server
// and pushed offline on the precondition that the receiver's application switches
// or the process is killed.
// The default title and content of offline push are kept.
chat.sendMessage(message);
```

```
chat.sendMessage(message, {
  // If the receiver is offline, the message will be neither stored on the roaming
  // nor pushed offline.
  onlineUserOnly: true
});
```

```
chat.sendMessage(message, {
  offlinePushInfo: {
    // If the receiver is offline, the message will be stored on the roaming server
    // but will not be pushed offline.
    disablePush: true
  }
});
```

```
chat.sendMessage(message, {
  offlinePushInfo: {
    title: '',
```

```
    description: '',
    androidInfo: {
       OPPOChannelID: ''
    }
  }
});
```

```
chat.sendMessage(message, {
  messageControlInfo: {
    // `unreadCount` of the conversation is not updated (the message is stored on t
    excludedFromUnreadCount: true,
    // `lastMessage` of the conversation is not updated (the message is stored on t
    excludedFromLastMessage: true
  }
});
```

```
chat.sendMessage(message, {
  messageControlInfo: {
    excludedFromContentModeration: true,
  }
});
```

```
// Support VoIP push
chat.sendMessage(message, {
  offlinePushInfo: {
    disablePush: false,
    disableVoipPush: false
  }
});
```

API-related Limits

Feature	Limit Item	Description
One-to-one/group message	Content length	A one-to-one or group message must be no longer than 12 KB.
	Sending frequency	One-to-one message: No limit for sending one-to-one messages on the client; sending through the RESTful API is subject to the frequency limit. See the corresponding API documentation. Group message: Up to 40 messages can be sent per second

		per group (regardless of the group type and platform interface, and this limit applies to each group separately).
	Receiving frequency	No limit for one-to-one messages or group messages.
	Size of a single file	SDKs support a maximum file size of 100 MB for any single file to be sent.

Note:

1. When the number of sent messages exceeds the limit, the backend will first deliver high-priority messages, and messages with the same priority will be delivered randomly. However, if the number of high-priority messages sent in a second exceeds 40, high-priority messages will also be discarded.
2. A message that has been restricted by frequency control is not delivered or stored in the message history, but a success response will be returned to the sender. Before Group Message Is Sent callback is triggered, but After a Group Message Is Sent callback is not triggered.
3. The default frequency limit for calling the RESTful API to send group messages is 200 times per second, which is a different concept from the aforementioned "limit of 40 messages per second for each group." Please distinguish it. Please refer to Use Limits for more limits.crc

©2013-2025 Tencent Cloud International Pte. Ltd.

Flutter

Last updated：2025-04-16 14:40:39

Overview

The method for sending a message is in the core class `TencentImSDKPlugin.v2TIMManager.getMessageManager()`.
It supports sending text, custom, and rich media messages, and all of which belong to the `V2TimMessage` type. `V2TimMessage` can contain different sub-types to indicate different types of messages.

Key APIs

The `sendMessage` (details) API is the core API for sending a message. It supports sending all types of messages.

Note:

The advanced message sending API mentioned below refers to `sendMessage`.

The API is detailed as follows:

```
Future<V2TimValueCallback<V2TimMessage>> sendMessage(
{
  required String id,
  required String receiver,
  required String groupID,
  int priority = 0,
  bool onlineUserOnly = false,
  bool needReadReceipt = false,
  bool isExcludedFromUnreadCount = false,
  bool isExcludedFromLastMessage = false,
  Map<String, dynamic>? offlinePushInfo,
  String? cloudCustomData,
  String? localCustomData,
}
)
```

Parameters:

Parameter	Definition	Valid for One-to-	Valid for Group Chat	Description

		One Chat		
id	ID returned after message creation	YES	YES	Create the message using the `createXxxMessage` API in advance.
receiver	`userID` of the one-to-one message receiver.	YES	NO	Just specify `receiver` for sending one-to-one chat messages.
groupID	`groupID` of the group chat	NO	YES	Just specify `groupID` for sending group messages.
priority	Message priority	NO	YES	Set a higher priority for important messages (such as those for red packets and gifts) and a lower priority for frequent and unimportant messages (such as those for likes).
onlineUserOnly	Whether only online users can receive the message.	YES	YES	If it is set to `true`, the message cannot be pulled by a receiver through historical messages. This is often used to implement weak prompts, such as "The other party is typing..." and unimportant prompts in a group.
offlinePushInfo	Offline push message	YES	YES	The title and content carried when a message is pushed offline.
needReadReceipt	Whether a read receipt is supported for the sent group message	NO	YES	Whether a read receipt is supported for the sent group message
isExcludedFromUnreadCount	Whether the sent message is counted as the unread message of the conversation	YES	YES	If it is set to `true`, the sent message is not counted as the unread message of the conversation. It defaults to `false`.
isExcludedFromLastMessage	Whether the sent message	YES	YES	If it is set to `true`, the sent message is not included in the `lastMessage` of

©2013-2025 Tencent Cloud International Pte. Ltd.

	is included in the `lastMessage` of the conversation			the conversation. It defaults to `false`.
cloudCustomData	Cloud message data	YES	YES	Extra data of the message that is stored in the cloud and can be accessed by the receiver.
localCustomData	Local message data	YES	YES	Extra data of the message that is stored locally. It cannot be accessed by the receiver and will be lost after the application is uninstalled.

Note:

If both `groupID` and `receiver` are set, targeted group messages are sent to `receiver`. For more information, see Targeted Group Message.

Sending Text Messages

Text messages include one-to-one messages and group messages, which are different in terms of APIs and parameters.

The ordinary and advanced APIs can be used to send text messages. The latter supports more sending parameters (such as priority and offline push message).

The ordinary API is as described below, while the advanced API is `sendMessage` mentioned above.

One-to-one text message

Advanced API

Take the following two steps to send a one-to-one text message with the advanced API:

1. Call `createTextMessage` (details) to create a text message.
2. Call `sendMessage` (details) to send the message.

Sample code:

```
// Create a text message
V2TimValueCallback<V2TimMsgCreateInfoResult> createTextMessageRes =
    await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .createTextMessage(
```

```
            text: "test", // Text message
          );
    if (createTextMessageRes.code == 0) {
      // Text message created successfully
      String? id = createTextMessageRes.data?.id;
      // Send the text message
      // During the call of `sendMessage`: If only `receiver` is entered, the messa
      // if only `groupID` is entered, the message is sent as a group message;
      // if `receiver` and `groupID` are entered, the message is sent to the specif
      V2TimValueCallback<V2TimMessage> sendMessageRes =
          await TencentImSDKPlugin.v2TIMManager.getMessageManager().sendMessage(
              id: id!, // ID of the created message
              receiver: "userID", // ID of the recipient
              groupID: "groupID", // ID of the receiving group
              priority: MessagePriorityEnum.V2TIM_PRIORITY_DEFAULT, // Message prio
              onlineUserOnly:
                  false, // Whether the message can be received only by online user
              isExcludedFromUnreadCount: false, // Whether the sent message is incl
              isExcludedFromLastMessage: false, // Whether the sent message is incl
              needReadReceipt:
                  false, // Whether the message requires a read receipt (to use thi
              offlinePushInfo: OfflinePushInfo(), // Title and content for offline
              cloudCustomData: "", // Cloud message data. Extra data of the message
              localCustomData:
                  "" // Local message data. Extra data of the message that is store
              );
      if (sendMessageRes.code == 0) {
        // Message sent successfully
      }
    }
```

Group text message

Advanced API

Take the following two steps to send a group text message with the advanced API:

1. Call `createTextMessage` (details) to create a text message.
2. Call `sendMessage` (details) to send the message.

Sample code:

```
    // Create a text message
    V2TimValueCallback<V2TimMsgCreateInfoResult> createTextMessageRes =
        await TencentImSDKPlugin.v2TIMManager
            .getMessageManager()
            .createTextMessage(
              text: "test", // Text message
```

```
            );
    if (createTextMessageRes.code == 0) {
      // Text message created successfully
      String? id = createTextMessageRes.data?.id;
      // Send the text message
      // During the call of `sendMessage`: If only `receiver` is entered, the messa
      // if only `groupID` is entered, the message is sent as a group message;
      // if `receiver` and `groupID` are entered, the message is sent to the specif
      V2TimValueCallback<V2TimMessage> sendMessageRes =
          await TencentImSDKPlugin.v2TIMManager.getMessageManager().sendMessage(
              id: id!, // ID of the created message
              receiver: "userID", // ID of the recipient
              groupID: "groupID", // ID of the receiving group
              priority: MessagePriorityEnum.V2TIM_PRIORITY_DEFAULT, // Message prio
              onlineUserOnly:
                  false, // Whether the message can be received only by online user
              isExcludedFromUnreadCount: false, // Whether the sent message is incl
              isExcludedFromLastMessage: false, // Whether the sent message is incl
              needReadReceipt:
                  false, // Whether the message requires a read receipt (to use thi
              offlinePushInfo: OfflinePushInfo(), // Title and content for offline
              cloudCustomData: "", // Cloud message data. Extra data of the message
              localCustomData:
                  "" // Local message data. Extra data of the message that is store
              );
      if (sendMessageRes.code == 0) {
        // Message sent successfully
      }
    }
```

Sending Custom Messages

Custom messages include one-to-one messages and group messages, which are different in terms of APIs and parameters. Custom messages can be sent with ordinary and advanced APIs.

The advanced API is `sendMessage` (details) mentioned above, which supports more sending parameters (such as priority and offline push message) than the ordinary API.

Custom one-to-one message

Advanced API

Take the following two steps to send a custom one-to-one message with the advanced API:

1. Call `createCustomMessage` (details) to create a custom message.

2. Call `sendMessage` (details) to send the message.

Sample code:

```
// Create a custom message
V2TimValueCallback<V2TimMsgCreateInfoResult> createCustomMessageRes = await Tencent
    data: 'Custom data',
    desc: 'Custom desc',
    extension: 'Custom extension',
  );
  if(createCustomMessageRes.code == 0){
    String id =  createCustomMessageRes.data.id;
    // Send the custom message
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin.v2TI
    if(sendMessageRes.code == 0){
      // Message sent successfully
    }
  }
```

Custom group message

Advanced API

Take the following two steps to send a custom group message with the advanced API:

1. Call `createCustomMessage` (details) to create a custom message.
2. Call `sendMessage` (details) to send the message.

Sample code:

```
// Create a custom message
V2TimValueCallback<V2TimMsgCreateInfoResult> createCustomMessageRes = await Tencent
    data: 'Custom data',
    desc: 'Custom desc',
    extension: 'Custom extension',
  );
  if(createCustomMessageRes.code == 0){
    String id =  createCustomMessageRes.data.id;
    // Send the custom message
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin.v2TI
    if(sendMessageRes.code == 0){
      // Message sent successfully
    }
  }
```

Sending Rich Media Messages

A rich media message can be sent only with the advanced API in the following steps:

1. Call `createXxxMessage` to create a rich media message object of a specified type, where `Xxx` indicates the specific message type.
2. Call `sendMessage` (details) to send the message.
3. Get the callback for message sending success or failure.

Image message

To create an image message, you need to get the path of a local image first.

During message sending, the image is uploaded to the server, and the upload progress is called back. The message is sent after the image is uploaded successfully.

If your project requires web support, the image sending mode is different from that on a mobile device. For details, see the web compatibility description section.

Sample code:

```
V2TimValueCallback<V2TimMsgCreateInfoResult> createImageMessageRes = await TencentI
        imagePath: "Absolute path of the local image",
);
  if (createImageMessageRes.code == 0) {
    String id = createImageMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
```

Audio message

To create an audio message, you need to get the path and audio duration of a local audio file first, where the audio duration is for display on the receiver UI.

During message sending, the audio is uploaded to the server, and the upload progress is called back. The message is sent after the audio is uploaded successfully.

Sample code:

```
V2TimValueCallback<V2TimMsgCreateInfoResult> createSoundMessageRes =
      await TencentImSDKPlugin.v2TIMManager.getMessageManager().createSoundMessage(
        soundPath: "Absolute path of the local audio file",
        duration: 10,// Audio duration
      );
  if (createSoundMessageRes.code == 0) {
    String id = createSoundMessageRes.data.id;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
```

Video message

To create a video message, you need to get the path, video duration, and video thumbnail of a local video file first, where the video duration and thumbnail are for display on the receiver UI.

During message sending, the video is uploaded to the server, and the upload progress is called back. The message is sent after the video is uploaded successfully.

If your project requires web support, the video sending mode is different from that on a mobile device. For details, see the web compatibility description section.

Sample code:

```
V2TimValueCallback<V2TimMsgCreateInfoResult> createVideoMessageRes =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createVideoMessage(
            videoFilePath: "Absolute path of the local video file",
            type: "mp4", // Video type
            duration: 10,// Video duration
            snapshotPath: "Absolute path of the local video thumbnail file",
          );
  if (createVideoMessageRes.code == 0) {
    String id = createVideoMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
```

File message

To create a file message, you need to get the path of a local file first.

During message sending, the file is uploaded to the server, and the upload progress is called back. The message is sent after the file is uploaded successfully.

If your project requires web support, the file sending mode is different from that on a mobile device. For details, see the web compatibility description section.

Sample code:

```
V2TimValueCallback<V2TimMsgCreateInfoResult> createFileMessageRes =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createFileMessage(
            filePath: "Absolute path of the local file",
            fileName: "File name",
          );
  if (createFileMessageRes.code == 0) {
    String id = createFileMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
```

Location message

Latitude and longitude information is sent in a location message, which requires a map control for display.

Sample code:

```
 V2TimValueCallback<V2TimMsgCreateInfoResult> createLocationMessage =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createLocationMessage(
            desc: "Shennan Boulevard, Nanshan District, Shenzhen",// Location infor
            longitude: 34,// Longitude
            latitude: 20, // Latitude
          );
  if (createLocationMessage.code == 0) {
    String id = createLocationMessage.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Emoji message

To send an emoji message, the corresponding emoji code is sent and then converted into the icon in the receiver.
Sample code:

```
V2TimValueCallback<V2TimMsgCreateInfoResult> createFaceMessageRes =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createFaceMessage(
            index: 0,
            data: "",
          );
  if (createFaceMessageRes.code == 0) {
    String id = createFaceMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
```

Support for the Flutter for Web

Chat SDK (tencent_cloud_chat_sdk) 4.1.1+2 or later version provides full compatibility with web. The steps for sending a media or file message in the web SDK are different from those of the Android SDK and iOS SDK:
Due to web characteristics, when you create a media or file message, the path cannot be directly passed in to the SDK. You need to get the DOM node for the input based on the element ID and pass in the input DOM after selecting the file.
For media selection, you are advised to use the image_picker package.
For file selection, you are advised to use the file_picker package.
If the value of `getElementById` in the sample code is different from the input ID that you see in the F12 console, the actual value prevails.

Sending an image

```
final ImagePicker _picker = ImagePicker();

_sendImageFileOnWeb() async {
  final pickedFile = await _picker.pickImage(source: ImageSource.gallery);
  final imageContent = await pickedFile!.readAsBytes();
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  fileName = pickedFile.name;
  tempFile = File(pickedFile.path);
  fileContent = imageContent;

  html.Node? inputElem;
  inputElem = html.document
      .getElementById("__image_picker_web-file-input")
      ?.querySelector("input");
  final convID = widget.conversationID;
  final convType =
  widget.conversationType == 1 ? ConvType.c2c : ConvType.group;
  final createImageMessageRes = await TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .createImageMessage(inputElement: inputElement);
  if (createImageMessageRes.code == 0) {
    String id = createImageMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
}
```

Sending a video

```
final ImagePicker _picker = ImagePicker();

_sendVideoFileOnWeb() async {
  final pickedFile = await _picker.pickVideo(source: ImageSource.gallery);
  final videoContent = await pickedFile!.readAsBytes();
  fileName = pickedFile.name ?? "";
  tempFile = File(pickedFile.path);
  fileContent = videoContent;

  if(fileName!.split(".")[fileName!.split(".").length - 1] != "mp4"){
    Toast.showToast("The video message must be in mp4 format.", context);
    return;
  }

  html.Node? inputElem;
  inputElem = html.document
      .getElementById("__image_picker_web-file-input")
      ?.querySelector("input");
```

```
  final convID = widget.conversationID;
  final convType =
  widget.conversationType == 1 ? ConvType.c2c : ConvType.group;
  final createVideoMessageRes = await TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .createVideoMessage(inputElement: inputElement, videoFilePath: "", type: "", du
  if (createVideoMessageRes.code == 0) {
    String id = createVideoMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
}
```

Sending a file

```
_sendFileOnWeb(){
  final convID = widget.conversationID;
  final convType =
      widget.conversationType == 1 ? ConvType.c2c : ConvType.group;
  FilePickerResult? result = await FilePicker.platform.pickFiles();
  if (result != null && result.files.isNotEmpty) {
    html.Node? inputElem;
    inputElem = html.document
        .getElementById("__file_picker_web-file-input")
        ?.querySelector("input");
    fileName = result.files.single.name;
    final createFileMessageRes = await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .createFileMessage(inputElement: inputElement, filePath: "", fileName: file
    if (createFileMessageRes.code == 0) {
    String id = createFileMessageRes.data.id;
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin
        .v2TIMManager
        .getMessageManager()
        .sendMessage(id: id, receiver: "userID", groupID: "groupID");
    if (sendMessageRes.code == 0) {
      // Message sent successfully
    }
  }
  }
}
```

Unity

Last updated：2024-01-31 15:50:48

Overview

You can send text, custom, and rich media messages, all of which belong to the `Message` type.

Key APIs

The `MsgSendMessage` API (details) is the core API for sending a message. It supports sending all types of messages.

The API is detailed as follows:

Type	Name	Description
System.String	conv_id	Conversation ID
TIMConvType	conv_type	Conversation type
Message	message	Message
System.Text.StringBuilder	message_id	Message ID
ValueCallback | ValueCallback	callback	Async callback

Sending Text Messages

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
```

```
            {
              elem_type = TIMElemType.kTIMElem_Text,
              text_elem_content =  "This is an ordinary text message"
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Sending Image Messages

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Image,
              image_elem_orig_path =  "/Users/xxx/xxx.png", // Absolute path to the
              image_elem_level = TIMImageLevel.kTIMImageLevel_Orig // Send an origi
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Sending a Voice Message

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Sound,
              sound_elem_file_path =  "/Users/xxx/xxx.mp3", // Absolute file path
              sound_elem_file_size = 10  // Audio duration

            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Sending a Video Message

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Video,
              video_elem_video_path =  "/Users/xxx/xxx.mp4", // Absolute file path
              video_elem_video_type = "mp4",  // Video type
              video_elem_video_duration = 10, // Video duration

              video_elem_image_path = "Absolute path of the local video thumbnail f
```

```
                video_elem_image_type = "png", // Type of the video screenshot file
                video_elem_image_size = 100, // Size of the video screenshot file
                video_elem_image_width = 1920, // Width of the video screenshot file
                video_elem_image_height = 1080, // Height of the video screenshot fil
              }
            }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Sending a File Message

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_File,
              file_elem_file_path =  "/Users/xxx/xxx.x", // Absolute file path
              file_elem_file_name = "Filename",
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Sending a Location Message

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Location,
              location_elem_desc =  "Shennan Boulevard, Nanshan District, Shenzhen"
              location_elem_longitude = 34, // Longitude
              location_elem_latitude = 20 // Latitude
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Send an Emoji Message

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Face,
              face_elem_index = 0,
              face_elem_buf = ""
```

```
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

Sending Custom Messages

```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Custom,
              custom_elem_data = "",
              custom_elem_desc = "",
              custom_elem_ext = ""
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128);

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
              // The message ID messageId returned when the message is sent
}
```

React Native

Last updated：2025-01-10 17:45:41

Creating a text message

This API is used to create a text message. It returns a message instance, which can be sent by calling the sendMessage API when you need to send a text message instance.

API

```
chat.createTextMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` (end-to-end conversation) or `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled
receiverList	Array | undefined	List of group members for targeted message reception (not supported for community and live broadcast groups)
isSupportExtension	Boolean	Whether message extension is supported, true for supported, false for not supported (you need to purchase the Pro edition 、 Pro Plus edition or Enterprise edition)

©2013-2025 Tencent Cloud International Pte. Ltd.

The description of `payload` is as follows:

Parameters	Type	Description
text	String	Message text content

Return Value

Message

Example

```
// Send text messages, same for Web and Mini Program
// 1. Create a message instance, the instance returned by the interface can be disp
let message = chat.createTextMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // Message priority for group chat. If the message frequency limit is exceeded in
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    text: 'Hello world!'
  },
  // If you need read receipts for your messages, you need to purchase the flagship
  needReadReceipt: true
  // Custom message data, saved in the cloud, will be sent to the peer end, and can
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send a message
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

```
// Sending a Targeted Group Message
// Note: Targeted group messages are excluded from the unread count of a conversati
let message = chat.createTextMessage({
  to: 'group1',
  conversationType: TencentCloudChat.TYPES.CONV_GROUP,
  payload: {
    text: 'Hello world!'
  },
```

```
  // To send a targeted group message, you must purchase the Enterprise Edition Pac
  receiverList: ['user0', 'user1']
});
// Send a message.
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Creating an @mention Message

TThis API is used to create a text message mentioning someone with @. It returns a message instance, which can be sent by calling the sendMessage API when you need to send a text message instance.

Notes:

1. This API is only for group chats, and @ALL is not supported in communities and topics under communities.
2. Creating group @ messages does not support specifying message recipients.

API

```
chat.createTextAtMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` (end-to-end conversation) or `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
text	String	Message text content
atUserList	Array	The list of users to @. If you need to @ALL, pass in TencentCloudChat.TYPES.MSG_AT_ALL. For example, if you want to @ notify both denny and lucy in this text message, and also @ everyone, pass ['denny', 'lucy', TencentCloudChat.TYPES.MSG_AT_ALL] to atUserList.

Return Value

Message

Example

```
// Send text messages, same for Web and Mini Program
// 1. Create a message instance, the instance returned by the interface can be disp
let message = chat.createTextAtMessage({
  to: 'group1',
  conversationType: TencentCloudChat.TYPES.CONV_GROUP,
  // Message priority for group chat
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    text: '@denny @lucy @everyone Dinner tonight, please reply 1 if received',
    // 'denny' and 'lucy' are userIDs, not nicknames
    atUserList: ['denny', 'lucy', TencentCloudChat.TYPES.MSG_AT_ALL]
  },
  // Custom message data, saved in the cloud, will be sent to the peer end, and can
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send a message
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Creating an image message

The API to create an image message returns a message instance, which can be sent using the send message API when you need to send an image message. For sending image messages in React Native, you need to integrate react-native-image-picker.

```
npm install react-native-image-picker --save
```

API

```
chat.createImageMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is described in the following table:

Parameters	Type	Description
to	String	Message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
onProgress	function	Callback function to get upload progress
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
file	HTMLInputElement | Object | File	DOM node for selecting images (Web) or File object (Web)

©2013-2025 Tencent Cloud International Pte. Ltd.

		The success callback parameter of the wx.chooseImage interface. The SDK will read the data and upload the image.

Return Value

Message

Example

```
import {launchImageLibrary} from 'react-native-image-picker';
// 1. Select an Image
launchImageLibrary({
  mediaType: 'photo',
  selectionLimit: 1,
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createImageMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Image
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // Sending succeeded.
    console.log(imResponse);
  }).catch(function(imError) {
    // Sending fails.
    console.warn('sendMessage error:', imError);
  });
});
```

```
import {launchCamera} from 'react-native-image-picker';
// 1. Take Photo
launchCamera({
  mediaType: 'photo',
  cameraType: 'back',
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createImageMessage({
    to: 'user1',
```

```
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Image
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // Sending succeeded.
    console.log(imResponse);
  }).catch(function(imError) {
    // Sending fails.
    console.warn('sendMessage error:', imError);
  });
});
```

Creating an Audio Message

The interface for creating an audio message instance. This interface returns a message instance, which can be used to call the Sending Message interface when you need to send an audio message. For sending audio messages in React Native, you need to integrate react-native-audio-recorder-player.

```
npm install react-native-audio-recorder-player --save
```

API

```
chat.createAudioMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	Message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_GROUP` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`

©2013-2025 Tencent Cloud International Pte. Ltd.

		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
file	Object	Audio file obtained after recording

Return Value

Message

Example

```
import AudioRecorderPlayer, {AVEncodingOption} from 'react-native-audio-recorder-pl

// Record the recording duration, unit: ms
let duration = 0;
// Record recording file path
let uri = '';

// 1. Start recording
const onStartRecord = async () => {
  await audioRecorderPlayer.startRecorder('test.aac',{
    VFormatIDKeyIOS: AVEncodingOption.aac,
  });
  audioRecorderPlayer.addRecordBackListener((e: any) => {
    duration = e.currentPosition;
  });
};
// 2. Stop recording
const onStopRecord = async () => {
  uri = await audioRecorderPlayer.stopRecorder();
  audioRecorderPlayer.removeRecordBackListener();
};
// 3. Send voice
const file = {
  uri: uri,
  duration: duration,
};
```

```
let message = chat.createAudioMessage({
  to: 'user1',
  conversationType: 'C2C',
  payload: {
    file: file,
  },
  // React Native does not support upload progress callback
  // onProgress: function(event) { console.log('file uploading:', event) }
});
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Creating a video message

The interface for creating a video message instance. This interface returns a message instance, which can be used to call the Sending Message interface when you need to send a video message. For sending video messages in React Native, you need to integrate react-native-image-picker.

```
npm install react-native-image-picker --save
```

API

```
chat.createVideoMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	Message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_GROUP` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`

		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL` `(default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of payload is as follows:

Parameters	Type	Description
file	HTMLInputElement | File | Object	Data fields of custom messages

Return Value

Message

Example

```
import {launchImageLibrary} from 'react-native-image-picker';
// 1. Select a Video
launchImageLibrary({
  mediaType: 'video',
  selectionLimit: 1,
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createVideoMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Video
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // Sending succeeded.
    console.log(imResponse);
  }).catch(function(imError) {
    // Sending fails.
```

```
    console.warn('sendMessage error:', imError);
  });
});
```

```
import {launchCamera} from 'react-native-image-picker';
// 1. Record Video
launchCamera({
  mediaType: 'video',
  cameraType: 'back',
}).then((result) => {
 const file = result.assets[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createVideoMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send Video
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // Sending succeeded.
    console.log(imResponse);
  }).catch(function(imError) {
    // Sending fails.
    console.warn('sendMessage error:', imError);
  });
});
```

Creating a custom message

This API is used to create a custom message instance. It returns a message instance, which can be sent by calling the sendMessage API when you need to send a custom message instance. When the capabilities provided by the SDK do not meet your needs, you can use custom messages for personalized customization.

API

```
chat.createCustomMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description

to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_GROUP` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
data	String	Data fields of custom messages
description	String	// Field that describes the custom message
extension	String	// Field that extends the custom message

Return Value

Message

Example

```
// Example: Implementing dice rolling using custom messages
// 1. Define a random function
function random(min, max) {
  return Math.floor(Math.random() * (max - min + 1) + min);
}
// 2. Create a message instance. The instance returned by the interface can be disp
let message = chat.createCustomMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // Message priority for group chat
```

```
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_HIGH,
  payload: {
    data: 'dice', // Identifies the message as a dice type message
    description: String(random(1,6)), // Get dice points
    extension: ''
  }
});
// 3. Send a message
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Creating an emoji message

This API is used to create an emoji message instance. It returns a message instance, which can be sent by calling the sendMessage API when you need to send an emoji message instance.

API

```
chat.createFaceMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

©2013-2025 Tencent Cloud International Pte. Ltd.

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
index	Number	Emoji index, user-defined
data	String	Extra data

Return Value

Message

Example

```
// Send emoji messages, same for Web and Mini Program.
// 1. Create a message instance, the instance returned by the interface can be disp
let message = chat.createFaceMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // Message priority for group chat
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    index: 1, // Number Emoji index, user-defined
    data: 'tt00' // String Extra data
  },
  // Custom message data, saved in the cloud, will be sent to the peer end, and can
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send a message
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Creating a file message

The interface for creating a file message. This interface returns a message instance, which can be used to call the Sending Message interface when you need to send a file message. For sending file messages in React Native, you need to integrate react-native-document-picker.

```
npm install react-native-document-picker --save
```

API

```
chat.createFileMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
onProgress	function	Callback function to get upload progress
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
file	HTMLInputElement | File | Object	The DOM node for selecting files (Web) or File object (Web) or Object (success callback parameter of the uni.chooseFile API). The SDK will read the data and upload the file.

©2013-2025 Tencent Cloud International Pte. Ltd.

Return Value

Message

Example

```
import DocumentPicker from 'react-native-document-picker';

// 1. Select File
DocumentPicker.pick({
  type: [DocumentPicker.types.allFiles],
}).then((result) => {
 const file = result[0];
 // 2. Create a message instance. The instance returned by the interface can be dis
 let message = chat.createFileMessage({
    to: 'user1',
    conversationType: TencentCloudChat.TYPES.CONV_C2C,
    payload: { file: file },
    // React Native does not support upload progress callback
    // onProgress: function(event) { console.log('file uploading:', event) }
  });
  // 3. Send File
  let promise = chat.sendMessage(message);
  promise.then(function(imResponse) {
    // Sending succeeded.
    console.log(imResponse);
  }).catch(function(imError) {
    // Sending fails.
    console.warn('sendMessage error:', imError);
  });
});
```

Creating a location message

This API is used to create a location message. It returns a message instance, which can be sent by calling the sendMessage API when you need to send a location message.

API

```
chat.createLocationMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description

to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
description	String	Location description information
longitude	Number	Longitude
latitude	Number	Latitude

Return Value

Message

Example

```
// Send location messages, same for Web and Mini Program
// 1. Create a message instance, the instance returned by the interface can be disp
let message = chat.createLocationMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // Message priority for group chat
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: {
    description: 'Tencent Building, 10000 Shennan Avenue, Shenzhen',
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    longitude: 113.941079, // Longitude
    latitude: 22.546103 // Latitude
  }
});
// 2. Send a message
let promise = chat.sendMessage(message);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Creating a Merged Message

This API is used to create a merged message. It returns a message instance, which can be sent by calling the sendMessage API when you need to send a merged message.

Notes:

1. Unable to merge messages that failed to be sent. If the message list contains a message that failed to be sent, the API will report an error.

API

```
chat.createMergerMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
to	String	The userID or groupID of the message recipient
conversationType	String	The conversation type, with values `TencentCloudChat.TYPES.CONV_C2C` or `TencentCloudChat.TYPES.CONV_GROUP`
priority	String	Message priority. If the message frequency limit is exceeded within the group, the backend will prioritize delivering high-priority messages. Supported enumeration values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)` `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

©2013-2025 Tencent Cloud International Pte. Ltd.

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the peer end, and can still be pulled after the application is uninstalled and reinstalled

The description of `payload` is as follows:

Parameters	Type	Description
messageList	Array	Merged message list
title	String	Title of merged messages, for example, "Chat History of the Talent Center in the Greater Bay Area"
abstractList	String	abstractList: Different message types can have different abstract information. For example, a text message can be set as: sender: text, an image message can be set as: sender: [Image], and a file message can be set as: sender: [File].
compatibleText	String	compatible text: If the lower version SDK does not support merged messages, a text message will be received by default. The content of the text message is ${compatibleText}, required.

Return Value

Message

Example

```
// 1. Forward group chat messages to c2c session
// message1 message2 message3 are group chat messages
let mergerMessage = chat.createMergerMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    messageList: [message1, message2, message3],
    title: 'Chat records of the Greater Bay Area Front-end Talent Center',
    abstractList: ['allen: 666', 'iris: [Image]', 'linda: [File]'],
    compatibleText: 'Upgrade your Chat SDK to v2.10.1 or later to view this message
  },
  // Custom message data, saved in the cloud, will be sent to the peer end, and can
  // cloudCustomData: 'your cloud custom data'
});

// 2. Send a message
```

```
let promise = chat.sendMessage(mergerMessage);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Downloading a Merged Message

This API is used to download a merged message. When the merged message sent by the sender is large in size, the SDK will store it in the cloud, and the message receiver needs to download it from the cloud before viewing it.

API

```
chat.downloadMergerMessage(message);
```

Parameters

Parameters	Type	Description
message	Message	Message instance

Return Value

`Promise`

Example

```
// The presence of downloadKey indicates that the received merged message is stored
if (message.type === TencentCloudChat.TYPES.MSG_MERGER && message.payload.downloadK
  let promise = chat.downloadMergerMessage(message);
  promise.then(function(imResponse) {
    // After the download is successful, the SDK will update information such as me
    console.log(imResponse.data);
  }).catch(function(imError) {
    // Download failed
    console.warn('downloadMergerMessage error:', imError);
  });
}
```

Forwarding Messages One by One

To forward a single message, create a message identical to the original message through the createForwardMessage API first, and then call the sendMessage API to send the message.

Notes:

1. It supports forwarding a single message or several messages one by one.

API

```
chat.createForwardMessage(message);
```

Parameters

Parameters	Type	Description
message	Message	Message instance

Example

```
let forwardMessage = chat.createForwardMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  // Message priority for group chat
  // priority: TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL,
  payload: message, // Message instance for the received or sent message
  // Custom message data, saved in the cloud, will be sent to the peer end, and can
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send a message
let promise = chat.sendMessage(forwardMessage);
promise.then(function(imResponse) {
  // Sending succeeded.
  console.log(imResponse);
}).catch(function(imError) {
  // Sending fails.
  console.warn('sendMessage error:', imError);
});
```

Sending Message

The API for sending messages requires you to first call the following APIs to create a message instance, and then call this API to send the message instance.

Create a text message

Create an @ message

Create an image message

Create an audio message

Create a video message

Create a custom message

Create an emoji message

Create a file message

Create a location message

Create a merged message

Create a forwarding message

Notes:

1. To send a message instance using this interface, the SDK must be in the ready state; otherwise, the message instance cannot be sent. The SDK state can be determined by listening to the following event: `TencentCloudChat.EVENT.SDK_READY` - triggered when the SDK is in the ready state.
2. `TencentCloudChat.EVENT.SDK_NOT_READY` - triggered when the SDK is in the not ready state.
3. To receive new messages for single chat, group chat, group tips, and group system notifications, listen to the event TencentCloudChat.EVENT.MESSAGE_RECEIVED.
4. Messages sent by this instance will not trigger the event TencentCloudChat.EVENT.MESSAGE_RECEIVED. Messages sent from the same account on other devices (or via REST API) will trigger the event TencentCloudChat.EVENT.MESSAGE_RECEIVED.
5. Offline push is only applicable to terminals (Android or iOS), Web is not supported.
6. onlineUserOnly and messageControlInfo cannot be used simultaneously.
7. Supports excluding unread counts for general community and topic messages.
8. Sending messages to [system conversation] will return error code 2106.
9. The message body size is limited to 12 KB and cannot be adjusted. Exceeding this limit will cause the message to fail to send, with error code 80002.
10. After enabling friend relationship check ([Chat Console] > [App Configuration] > [Feature Configuration] > [Login and Messages] > [Friend Relationship Check]), Chat will check the friend relationship when initiating a one-on-one chat, allowing only friends to send one-on-one messages. When a stranger sends a one-on-one message, the SDK returns error code 20009.

```
chat.sendMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. The attribute it contains is as follows:

Parameters	Type	Description
message	Message	Message instance
options	Object	Message sending options (message content container), optional

The description of `options` is as follows:

Parameters	Type	Description
onlineUserOnly	Boolean	Indicator of whether the message is sent only to online users, default value is false; if set to true, the message will neither be stored in roaming nor counted as unread, and will not be pushed offline to the recipient. Suitable for scenarios such as sending broadcast notifications or other unimportant prompt messages. This option is not supported when sending messages in AVChatRoom.
offlinePushInfo	Object	Offline Push configuration
messageControlInfo	Object	Message control configuration

The description of `offlinePushInfo` is as follows:

Parameters	Type	Description
disablePush	Boolean	true to disable offline push; false to enable offline push (default)
disableVoipPush	Boolean	true to disable voip push (default); false to enable voip push (enabling voip push requires offline push to be enabled)
title	String	Offline push title, this field is shared by iOS and Android
description	String	Offline push content. This field will override the offline push display text of the message instance. If a custom message is sent, this description field will override message.payload.description. If neither description nor message.payload.description is filled, the recipient will not receive the offline push of this custom message.
extension	String	Offline push pass-through content
androidInfo	Object	Android push configuration
apnsInfo	Object	iOS push configuration

The description of `androidInfo` is as follows:

Parameters	Type	Description

©2013-2025 Tencent Cloud International Pte. Ltd.

sound	String	Android offline push sound settings. Only supported by Huawei, Xiaomi, and Google. For Xiaomi phones with Android 8.0 and above, you must set androidInfo.XiaoMiChannelID. Please refer to: Custom ringtone for Xiaomi. Google Phone FCM Push on Android 8.0 and above systems requires setting androidInfo.FCMChannelID for sound notifications. Custom ringtone requires setting the sound file path. For example: androidInfo.sound = "shake.xxx" Corresponding to the uniapp "nativeResources/android/res/raw/shake.xxx" audio file. Corresponding to the native application's "/res/raw/shake.xxx" audio file.
XiaoMiChannelID	String	Notification category (Channel) adaptation field for Xiaomi phones with MIUI 10 and above.
OPPOChannelID	String	NotificationChannel adaptation fields for OPPO phones with Android 8.0 and above.
FCMChannelID	String	Notification channel fields for Google phones with Android 8.0 and above.
VIVOClassification	Number	VIVO phone push message classification, 0: operational message, 1: system message, default is 1.
VIVOCategory	String	Used to identify message type on VIVO phones.
HuaWeiCategory	String	Used to identify message type on Huawei phones.
HuaWeiImage	String	Huawei push notification bar notification image URL, supported from v3.4.1. The protocol used must be HTTPS, example value: https://example.com/image.png Image files must be smaller than 512 KB, with recommended dimensions of 40dp x 40dp and a corner radius of 8dp. Images exceeding the recommended specifications may be compressed or not fully displayed. Recommended formats are JPG/JPEG/PNG.
HonorImage	String	Honor push notification bar notification image URL, supported from v3.4.1. The protocol used must be HTTPS, example value: https://example.com/image.png

©2013-2025 Tencent Cloud International Pte. Ltd.

		Icon files must be smaller than 100 KB, with recommended dimensions of 160px x 160px and a corner radius of 32px. Icons exceeding the recommended specifications may be compressed or not fully displayed.
GoogleImage	String	Google push notification bar notification image URL, supported from v3.4.1. The protocol used must be HTTPS, example value: https://example.com/image.png Icon files must be smaller than 1 MB. Icons exceeding the recommended specifications may be compressed or not fully displayed.

The description of `apnsInfo` is as follows:

Parameters	Type	Description
sound	String	iOS offline push sound settings. When apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_NO_SOUND, it means no sound will be played upon receiving. When apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_DEFAULT_SOUND, it means the system sound will be played upon receiving. Custom ringtone requires setting the sound file path. For example: apnsInfo.sound = "shake.xxx" Corresponding to the uniapp "nativeResources/ios/Resources/shake.xxx" file, uniapp must be a release package. Corresponding to the native Xcode project's "shake.xxx" audio file.
ignoreIOSBadge	Boolean	Offline push ignores badge count (effective only for iOS), default is false. When set to true, this message will not increase the app icon's unread count on the iOS receiving end.
disableVoipPush	Boolean	true to disable voip push (default); false to enable voip push (enabling voip push requires offline push to be enabled)
image	String	Identifies the APNs notification bar notification image URL. When the client receives this field, it can display the image in the popup by downloading the image resource. Supported from v3.4.1. The protocol used must be HTTPS, example value: https://example.com/image.png Supports JPEG, GIF, PNG, size not exceeding 10 MB. The mutable-content attribute needs to be enabled in the IM console to support iOS 10 Service Extension features.

©2013-2025 Tencent Cloud International Pte. Ltd.

		The key value of the resource is "image".

The description of `messageControlInfo` is as follows:

Parameters	Type	Description
excludedFromUnreadCount	Boolean	true: The message does not update the conversation unreadCount (message stored in roaming), default value is false.
excludedFromLastMessage	Boolean	true: The message does not update the conversation lastMessage (message stored in roaming), default value is false.
excludedFromContentModeration	Boolean	Whether the message bypasses content moderation (cloud moderation). The excludedFromContentModeration setting is only effective after enabling the [cloud moderation] feature. Set to true to bypass content moderation, set to false to pass content moderation.

Return

`Promise`

Example

```
// If the recipient is offline, the message will be stored for roaming and an offli
// The title and content of the offline push use default values.
// For details on offline push, please refer to https://www.tencentcloud.com/docume
chat.sendMessage(message);
```

```
chat.sendMessage(message, {
  onlineUserOnly: true // If the recipient is offline, the message will not be stor
});
```

```
chat.sendMessage(message, {
  offlinePushInfo: {
    disablePush: true // If the recipient is offline, the message will be stored fo
  }
});
```

```
chat.sendMessage(message, {
  // If the recipient is offline, the message will be stored for roaming and an off
```

```
  // The access side can customize the title and content of offline push
  offlinePushInfo: {
    title: '', // Offline push title
    description: '', // Offline push content
    androidOPPOChannelID: '' // Offline push setting for OPPO phones with system 8.
  }
});
```

```
chat.sendMessage(message, {
  messageControlInfo: {
    excludedFromUnreadCount: true, // The message does not update the conversation
    excludedFromLastMessage: true // The message does not update the conversation l
  }
});
```

```
// The message bypasses content moderation
chat.sendMessage(message, {
  messageControlInfo: {
    excludedFromContentModeration: true,
  }
});
```

```
// Supports VoIP push
chat.sendMessage(message, {
  offlinePushInfo: {
    disablePush: false,
    disableVoipPush: false
  }
});
```

Getting Device Permissions

React Native requires relevant device permissions to send image, video, and file messages.

Android

iOS

In the android directory, find app/src/main/AndroidManifest.xml and add the following permissions:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

In the ios directory, find Info.plist and add the following permissions:

```
<key>NSCameraUsageDescription</key>
<string>We need access to your camera to take photos</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>We need access to your album to select photos</string>
<key>NSMicrophoneUsageDescription</key>
<string>We need access to your microphone to record audio</string>
```

Interface Limitations

Features	Restriction Item	Explanation
One-to-one chat/Group chat	Content length	The maximum length of a single one-to-one or group chat message is 12K.
	Sending frequency	One-to-one chat messages: No limit for client-sent messages; REST API sent messages have frequency limits, refer to the respective interface documentation. Group chat messages: Limited to 40 messages per second per group (applies to all group types and all platform interfaces). Frequency limits in different groups do not affect each other.
	Receiving frequency	No limit for both one-to-one and group chats.
	Individual File Size	When sending file messages, the SDK supports a maximum individual file size of 100 MB.

Notes:

1. When the number of messages exceeds the limit, the backend prioritizes messages with relatively higher priority. Messages with the same priority are sorted randomly. If the number of high-priority messages sent in a second exceeds 40, high-priority messages will also be discarded.
2. A message that has been restricted by frequency control is not delivered or stored in the message history, but a success response will be returned to the sender. Before Group Message Is Sent callback is triggered, but After a Group Message Is Sent callback is not triggered.
3. The default frequency limit for calling the REST API to send group messages is 200 times/second, which is different from the aforementioned "40 messages/second per group" concept. Please distinguish between them.

For more restrictions, see the document Limits.

©2013-2025 Tencent Cloud International Pte. Ltd.

Receiving Message Android&iOS&Windows&Mac

Last updated：2025-01-13 17:27:06

Feature Description

The `addSimpleMsgListener` API is used to listen for and receive text and custom messages, with the callback defined in the `V2TIMSimpleMsgListener` protocol.
The `addAdvancedMsgListener` API is used to listen for and receive all types of messages (text, custom, and rich media messages), with the callback defined in the `V2TIMAdvancedMsgListener` protocol.

Setting a Message Listener

The SDK provides the `V2TIMSimpleMsgListener` simple message listener and the `V2TIMAdvancedMsgListener` advanced message listener.
They differ in that:
1. The simple message listener **can only** receive text and custom messages. You can use it if your business only requires these two types of messages.
2. The advanced message listener can receive **all** types of messages. You can use it if your business also requires rich media, merged, and other messages.
Note:
1. `addSimpleMsgListener` and `addAdvancedMsgListener` are exclusive. **Do not use both of them**; otherwise, unpredictable logic bugs will occur.
2. A message listener **must** be added first to receive the following types of messages.

Simple message listener

Adding a listener

The receiver calls the `addSimpleMsgListener` (Java / Swift / Objective-C / C++) to add the simple message listener. We recommend it be called early, such as after the chat page is initialized, to ensure timely message receiving in the application.
Sample code:
Java
Swift
Objective-C

©2013-2025 Tencent Cloud International Pte. Ltd.

C++

```
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);
```

```
V2TIMManager.shared.addSimpleMsgListener(listener: self)
```

```
// `self` is id<V2TIMSignalingListener>
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];
```

```
class SimpleMsgListener final : public V2TIMSimpleMsgListener {
    // Members ...
};

// Note that `simpleMsgListener` should not be released before the IM SDK is uninit
// otherwise the message callback cannot be called.
SimpleMsgListener simpleMsgListener;
V2TIMManager::GetInstance()->AddSimpleMsgListener(&simpleMsgListener);
```

Listener callback event

After adding the simple message listener, the receiver can receive different types of messages in the callback of `V2TIMSimpleMsgListener` (Java / Swift / Objective-C / C++), as described below:

Java

Swift

Objective-C

C++

```
public abstract class V2TIMSimpleMsgListener {
    // Received the one-to-one text message
    public void onRecvC2CTextMessage(String msgID, V2TIMUserInfo sender, String tex

    // Received the custom one-to-one (signaling) message
    public void onRecvC2CCustomMessage(String msgID, V2TIMUserInfo sender, byte[] c

    // Received the group text message
    public void onRecvGroupTextMessage(String msgID, String groupID, V2TIMGroupMemb

    // Received the custom group (signaling) message
    public void onRecvGroupCustomMessage(String msgID, String groupID, V2TIMGroupMe
}
```

```
public protocol V2TIMSimpleMsgListener : AnyObject {
    // Received the one-to-one text message
```

```
    func onRecvC2CTextMessage(msgID: String, sender: V2TIMUserInfo, text: String?)
    // Received the custom one-to-one (signaling) message
    func onRecvC2CCustomMessage(msgID: String, sender: V2TIMUserInfo, customData: D
    // Received the group text message
    func onRecvGroupTextMessage(msgID: String, groupID: String, sender: V2TIMGroupM
    // Received the custom group (signaling) message
    func onRecvGroupCustomMessage(msgID: String, groupID: String, sender: V2TIMGrou

}
```

```
/// Basic message callback of the IM SDK
@protocol V2TIMSimpleMsgListener <NSObject>
@optional

/// Received a one-to-one text message
- (void)onRecvC2CTextMessage:(NSString *)msgID  sender:(V2TIMUserInfo *)info text:(

/// Received a custom one-to-one (signaling) message
- (void)onRecvC2CCustomMessage:(NSString *)msgID  sender:(V2TIMUserInfo *)info cust

/// Received a group text message
- (void)onRecvGroupTextMessage:(NSString *)msgID groupID:(NSString *)groupID sender

/// Received a custom group (signaling) message
- (void)onRecvGroupCustomMessage:(NSString *)msgID groupID:(NSString *)groupID send
@end
```

```
class SimpleMsgListener final : public V2TIMSimpleMsgListener {
public:
    SimpleMsgListener() = default;
    ~SimpleMsgListener() override = default;

    // Received the one-to-one text message
    void OnRecvC2CTextMessage(const V2TIMString& msgID, const V2TIMUserFullInfo& se
                              const V2TIMString& text) override {}

    // Received the custom one-to-one (signaling) message
    void OnRecvC2CCustomMessage(const V2TIMString& msgID, const V2TIMUserFullInfo&
                                const V2TIMBuffer& customData) override {}

    // Received the group text message
    void OnRecvGroupTextMessage(const V2TIMString& msgID, const V2TIMString& groupI
                                const V2TIMGroupMemberFullInfo& sender, const V2TIM

    // Received the custom group (signaling) message
```

```
    void OnRecvGroupCustomMessage(const V2TIMString& msgID, const V2TIMString& grou
                                  const V2TIMGroupMemberFullInfo& sender,
                                  const V2TIMBuffer& customData) override {}
};
```

Removing a listener

To stop receiving messages, the receiver can call `removeSimpleMsgListener` (Java / Swift / Objective-C / C++) to remove the simple message listener.

Sample code:

Java

Swift

Objective-C

C++

```
V2TIMManager.getInstance().removeSimpleMsgListener(simpleMsgListener);
```

```
V2TIMManager.shared.removeSimpleMsgListener(listener: self)
```

```
// `self` is id<V2TIMSignalingListener>
[[V2TIMManager sharedInstance] removeSimpleMsgListener:self];
```

```
class SimpleMsgListener final : public V2TIMSimpleMsgListener {
    // Members ...
};

// `simpleMsgListener` is the instance of SimpleMsgListener
V2TIMManager::GetInstance()->RemoveSimpleMsgListener(&simpleMsgListener);
```

Advanced message listener

Adding a listener

The receiver calls the `addAdvancedMsgListener` (Java / Swift / Objective-C / C++) to add the advanced message listener. We recommend it be called early, such as after the chat page is initialized, to ensure timely message receiving in the application.

Sample code:

Java

Swift

Objective-C

C++

©2013-2025 Tencent Cloud International Pte. Ltd.

```
V2TIMManager.getMessageManager().addAdvancedMsgListener(advancedMsgListener);
```

```
V2TIMManager.shared.addAdvancedMsgListener(listener: self)
```

```
// `self` is id<V2TIMAdvancedMsgListener>
[[V2TIMManager sharedInstance] addAdvancedMsgListener:self];
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
    // Members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Listener callback event

After adding the advanced message listener, the receiver can receive different types of messages in the callback of `V2TIMAdvancedMsgListener` (Java / Swift / Objective-C / C++), as described below:

Java

Swift

Objective-C

C++

```
public abstract class V2TIMAdvancedMsgListener {
    // Received a new message
    public void onRecvNewMessage(V2TIMMessage msg) {}

    // one-to-one message read notification (a notification is received when the re
    public void onRecvC2CReadReceipt(List<V2TIMMessageReceipt> receiptList) {}

    // Message read receipt notification (if read receipts are supported, the notif
    public void onRecvMessageReadReceipts(List<V2TIMMessageReceipt> receiptList) {}

    // Received a message recall notification
    public void onRecvMessageRevoked(String msgID) {}

    // The message content is modified.
    public void onRecvMessageModified(V2TIMMessage msg) {}
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
public protocol V2TIMAdvancedMsgListener: AnyObject {
    // Received a new message
    func onRecvNewMessage(msg: V2TIMMessage)

    // Message read receipt notification (if read receipts are supported, the notif

    func onRecvMessageReadReceipts(receiptList: Array<V2TIMMessageReceipt>)

    // one-to-one message read notification (a notification is received when the re

    func onRecvC2CReadReceipt(receiptList: Array<V2TIMMessageReceipt>)

    // Received a message recall notification
    func onRecvMessageRevoked(msgID: String, operateUser: V2TIMUserInfo, reason: St

    // The message content is modified.
    func onRecvMessageModified(msg: V2TIMMessage)
}
```

```
/// Advanced message listener
@protocol V2TIMAdvancedMsgListener <NSObject>
@optional
/// Received a new message
- (void)onRecvNewMessage:(V2TIMMessage *)msg;

/// Message read receipt notification (if read receipts are supported, the notifica
- (void)onRecvMessageReadReceipts:(NSArray<V2TIMMessageReceipt *> *)receiptList;

/// One-to-one message read notification (a notification is received when the recei
- (void)onRecvC2CReadReceipt:(NSArray<V2TIMMessageReceipt *> *)receiptList;

/// Received a message recall notification
- (void)onRecvMessageRevoked:(NSString *)msgID;

/// The message content is modified
- (void)onRecvMessageModified:(V2TIMMessage *)msg;
@end
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    AdvancedMsgListener() = default;
    ~AdvancedMsgListener() override = default;

    // Received a new message
    void OnRecvNewMessage(const V2TIMMessage& message) override {}
```

```
    // one-to-one message read notification (a notification is received when the re
    void OnRecvC2CReadReceipt(const V2TIMMessageReceiptVector& receiptList) overrid

    // Message read receipt notification (if read receipts are supported, the notif
    void OnRecvMessageReadReceipts(const V2TIMMessageReceiptVector& receiptList) ov

    // Received a message recall notification
    void OnRecvMessageRevoked(const V2TIMString& messageID) override {}

    // The message content is modified.
    void OnRecvMessageModified(const V2TIMMessage& message) override {}
};
```

Removing a listener

To stop receiving messages, the receiver can call `removeAdvancedMsgListener` (Java / Swift / Objective-C / C++) to remove the advanced message listener.

Sample code:

Java

Swift

Objective-C

C++

```
V2TIMManager.getMessageManager().removeAdvancedMsgListener(advancedMsgListener);
```

```
V2TIMManager.shared.removeAdvancedMsgListener(listener: self)
```

```
// `self` is id<V2TIMAdvancedMsgListener>
[[V2TIMManager sharedInstance] removeAdvancedMsgListener:self];
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
    // Members ...
};

// `advancedMsgListener` is the instance of AdvancedMsgListener
V2TIMManager::GetInstance()->GetMessageManager()->RemoveAdvancedMsgListener(&advanc
```

Receiving a Text Message

Using simple message listener

One-to-one text message

The receiver can receive a one-to-one text message by using the simple message listener in the following steps:

1. Call `addSimpleMsgListener` to set the event listener.
2. Listen for the `onRecvC2CTextMessage` callback (Java / Swift / Objective-C / C++) to receive text messages.
3. To stop receiving messages, call `removeSimpleMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

Java

Swift

Objective-C

C++

```
// Set the event listener
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

// Receive the one-to-one text message
/**
 * Received the one-to-one text message
 *
 * @param msgID Unique message ID
 * @param sender Sender information
 * @param text The sent content
 */
public void onRecvC2CTextMessage(String msgID, V2TIMUserInfo sender, String text) {
    // Parse the message and display it on the UI
}
```

```
// Set the event listener
V2TIMManager.shared.addSimpleMsgListener(listener: self)

// Receive the one-to-one text message
/**
 * Received the one-to-one text message
 *
 * @param msgID Unique message ID
 * @param sender Sender information
 * @param text The sent content
 */
func onRecvC2CTextMessage(msgID: String, sender: V2TIMUserInfo, text: String?) {
    // Parse the message and display it on the UI
}
```

```
// Set the event listener
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];

/// Receive the one-to-one text message
/// @param msgID Message ID
/// @param info Sender information
/// @param text Text content
- (void)onRecvC2CTextMessage:(NSString *)msgID sender:(V2TIMUserInfo *)info text:(N
    // Parse the message and display it on the UI
}
```

```
class SimpleMsgListener final : public V2TIMSimpleMsgListener {
public:
    /**
     * Receive the one-to-one text message
     *
     * @param msgID Unique message ID
     * @param sender Sender information
     * @param text The sent content
     */
    void OnRecvC2CTextMessage(const V2TIMString& msgID, const V2TIMUserFullInfo& se
                              const V2TIMString& text) override {
        // Parse the message and display it on the UI
        std::cout << "text:" << std::string{text.CString(), text.Size()} << std::en
    }
    // Other members ...
};

// Note that `simpleMsgListener` should not be released before the IM SDK is uninit
// otherwise the message callback cannot be called.
SimpleMsgListener simpleMsgListener;
V2TIMManager::GetInstance()->AddSimpleMsgListener(&simpleMsgListener);
```

Group text message

The receiver can receive a group text message by using the simple message listener in the following steps:

1. Call `addSimpleMsgListener` to set the event listener.
2. Listen for the `onRecvGroupTextMessage` callback (Java / Swift / Objective-C / C++) to receive text messages.
3. To stop receiving messages, call `removeSimpleMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

Java

Swift

Objective-C

C++

```
// Set the event listener
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

// Receive the group text message
/**
 * Received the group text message
 *
 * @param msgID Unique message ID
 * @param groupID Group ID
 * @param sender The group member information of the sender
 * @param text The sent content
 */
public void onRecvGroupTextMessage(String msgID, String groupID, V2TIMGroupMemberIn
    // Parse the message and display it on the UI
}
```

```
// Set the event listener
V2TIMManager.shared.addSimpleMsgListener(listener: self)
// Receive the group text message
/**
 * Received the group text message
 *
 * @param msgID Unique message ID
 * @param groupID Group ID
 * @param sender The group member information of the sender
 * @param text The sent content
 */
func onRecvGroupTextMessage(msgID: String, groupID: String, sender: V2TIMGroupMembe
 {
    // Parse the message and display it on the UI
}
```

```
// Set the event listener
[[V2TIMManager sharedInstance] addSimpleMsgListener:self];

/// Receive the group text message
/// @param msgID Message ID
/// @param groupID Group ID
/// @param info Sender information
/// @param text Text content
- (void)onRecvGroupTextMessage:(NSString *)msgID groupID:(NSString *)groupID sender
```

```
    // Parse the message and display it on the UI
}
```

```
class SimpleMsgListener final : public V2TIMSimpleMsgListener {
public:
    /**
     * Received the group text message
     *
     * @param msgID Unique message ID
     * @param groupID Group ID
     * @param sender The group member information of the sender
     * @param text The sent content
     */
    void OnRecvGroupTextMessage(const V2TIMString& msgID, const V2TIMString& groupI
                                const V2TIMGroupMemberFullInfo& sender, const V2TIM
        // Parse the message and display it on the UI
        std::cout << "text:" << std::string{text.CString(), text.Size()} << std::en
    }
    // Other members ...
};

// Note that `simpleMsgListener` should not be released before the IM SDK is uninit
// otherwise the message callback cannot be called.
SimpleMsgListener simpleMsgListener;
V2TIMManager::GetInstance()->AddSimpleMsgListener(&simpleMsgListener);
```

Using advanced message listener

The receiver can receive a one-to-one or group text message by using the advanced message listener in the following steps:

1. Call `addAdvancedMsgListener` to set the event listener.
2. Listen for the `onRecvNewMessage` callback (Java / Swift / Objective-C / C++) to receive text messages.
3. To stop receiving messages, call `removeAdvancedMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

Java

Swift

Objective-C

C++

```
// Set the event listener
V2TIMManager.getMessageManager().addAdvancedMsgListener(advancedMsgListener);

/**
```

```
 * Received a new message
 * @param msg Message
 */
public void onRecvNewMessage(V2TIMMessage msg) {
    // Parse the `groupID` and `userID`
    String groupID = msg.getGroupID();
    String userID = msg.getUserID();

    // Determine whether it's a one-to-one chat or group chat
    // If `groupID` is not empty, the message is from a group chat; if `userID` is

    // Parse the text message in `msg`
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_TEXT) {
        V2TIMTextElem textElem = msg.getTextElem();
        String text = textElem.getText();
        Log.i("onRecvNewMessage", "text:" + text);
    }
}
```

```
// Set the event listener
V2TIMManager.shared.addAdvancedMsgListener(listener: self)

/**
 * Received a new message
 * @param msg Message
 */
func onRecvNewMessage(_ msg: V2TIMMessage) {
    // Parse the `groupID` and `userID`
    let groupID = msg.groupID
    let userID = msg.userID

   // Determine whether it's a one-to-one chat or group chat
    // If `groupID` is not empty, the message is from a group chat; if `userID` is
    if !groupID.isEmpty {
        print("Received a group message in group: \\(groupID)")
    } else if !userID.isEmpty {
        print("Received a one-on-one message from user: \\(userID)")
    } else {
        print("Received a message with no identifiable sender.")
    }

    // Parse the text message in `msg`
    if msg.elemType == .V2TIM_ELEM_TYPE_TEXT {
        if let textElem = msg.textElem {
            let text = textElem.text
            print("onRecvNewMessage, text: \\(text)")
```

```
        }
    } else {
        print("Received a non-text message.")
    }
}
```

```
// Set the event listener
[[V2TIMManager sharedInstance] addAdvancedMsgListener:self];

/// Receive the message
/// @param msg Message object
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    // Parse the `groupID` and `userID`
    NSString *groupID = msg.groupID;
    NSString *userID = msg.userID;

    // Determine whether it's a one-to-one chat or group chat
    // If `groupID` is not empty, the message is from a group chat; if `userID` is

    // Parse the text message in `msg`
    if (msg.elemType == V2TIM_ELEM_TYPE_TEXT) {
        V2TIMTextElem *textElem = msg.textElem;
        NSString *text = textElem.text;
        NSLog(@"onRecvNewMessage, text: %@", text);
    }
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    /**
     * Received a new message
     *
     * @param message Message
     */
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        // Parse the `groupID` and `userID`
        V2TIMString groupID = message.groupID;
        V2TIMString userID = message.userID;

        // Determine whether it's a one-to-one chat or group chat
        // If `groupID` is not empty, the message is from a group chat;
        // if `userID` is not empty, the message is from a one-to-one chat.

        // Parse the text message in `message`
        if (message.elemList.Size() == 1) {
```

```
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_TEXT) {
                auto textElem = static_cast<V2TIMTextElem*>(elem);
                V2TIMString text = textElem->text;
                // Parse the message and display it on the UI
                std::cout << "text:" << std::string{text.CString(), text.Size()} <<
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Receiving a Custom Message

Using simple message listener

Custom one-to-one message

The receiver can receive a custom one-to-one message by using the simple message listener in the following steps:

1. Call `addSimpleMsgListener` to set the event listener.
2. Listen for the `onRecvC2CCustomMessage` callback (Java / Swift / Objective-C / C++) to receive custom one-to-one messages.
3. To stop receiving messages, call `removeSimpleMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

Java

Swift

Objective-C

C++

```
/**
 * Receive the custom one-to-one message
 * @param msgID Message ID
 * @param sender Sender information
 * @param customData The sent content
 */
public void onRecvC2CCustomMessage(String msgID, V2TIMUserInfo sender, byte[] custo
```

```
    Log.i("onRecvC2CCustomMessage", "msgID:" + msgID + ", from:" + sender.getNickNa
}
```

```
/**
 * Receive the custom one-to-one message
 * @param msgID Message ID
 * @param sender Sender information
 * @param customData The sent content
 */
func onRecvC2CCustomMessage(msgID: String, sender: V2TIMUserInfo, customData: Data?

}
```

```
/// Receive the custom one-to-one message
/// @param msgID Message ID
/// @param info Sender information
/// @param data The binary content of the custom message
- (void)onRecvC2CCustomMessage:(NSString *)msgID sender:(V2TIMUserInfo *)info custo
    NSLog(@"onRecvC2CCustomMessage, msgID: %@, sender: %@, customData: %@", msgID,
}
```

```
class SimpleMsgListener final : public V2TIMSimpleMsgListener {
public:
    /**
     * Receive the custom one-to-one message
     *
     * @param msgID Message ID
     * @param sender Sender information
     * @param customData The sent content
     */
    void OnRecvC2CCustomMessage(const V2TIMString& msgID, const V2TIMUserFullInfo&
                                const V2TIMBuffer& customData) override {
        // Parse the message and display it on the UI
        std::cout << "customData:"
                  << std::string{reinterpret_cast<const char*>(customData.Data()),
                  << std::endl;
    }
    // Other members ...
};

// Note that `simpleMsgListener` should not be released before the IM SDK is uninit
// otherwise the message callback cannot be called.
SimpleMsgListener simpleMsgListener;
V2TIMManager::GetInstance()->AddSimpleMsgListener(&simpleMsgListener);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Custom group message

The receiver can receive a custom group message by using the simple message listener in the following steps:

1. Call `addSimpleMsgListener` to set the event listener.
2. Listen for the `onRecvGroupCustomMessage` callback (Java / Swift / Objective-C / C++) to receive custom group messages.
3. To stop receiving messages, call `removeSimpleMsgListener` to remove the listener. This step is optional and can be performed as needed.

Java

Swift

Objective-C

C++

```
/**
 * Receive the custom group message
 * @param msgID Message ID
 * @param groupID Group ID
 * @param sender The group member information of the sender
 * @param customData The sent content
 */
public void onRecvGroupCustomMessage(String msgID, String groupID, V2TIMGroupMember
    Log.i("onRecvGroupCustomMessage", "msgID:" + msgID + ", groupID:" + groupID + "
}
```

```
/**
 * Receive the custom group message
 * @param msgID Message ID
 * @param groupID Group ID
 * @param sender The group member information of the sender
 * @param customData The sent content
 */
func onRecvGroupCustomMessage(msgID: String, groupID: String, sender: V2TIMGroupMem

}
```

```
/// Receive the custom group message
/// @param msgID Message ID
/// @param groupID Group ID
/// @param info Sender information
/// @param text The binary content of the custom message
- (void)onRecvGroupCustomMessage:(NSString *)msgID groupID:(NSString *)groupID send
    NSLog(@"onRecvGroupCustomMessage, msgID: %@, groupID: %@, sender: %@, customDat
```

```
}

class SimpleMsgListener final : public V2TIMSimpleMsgListener {
public:
    /**
     * Receive the custom group message
     *
     * @param msgID Message ID
     * @param groupID Group ID
     * @param sender The group member information of the sender
     * @param customData The sent content
     */
    void OnRecvGroupCustomMessage(const V2TIMString& msgID, const V2TIMString& grou
                                  const V2TIMGroupMemberFullInfo& sender,
                                  const V2TIMBuffer& customData) override {
        // Parse the message and display it on the UI
        std::cout << "customData:"
                  << std::string{reinterpret_cast<const char*>(customData.Data()),
                  << std::endl;
    }
    // Other members ...
};

// Note that `simpleMsgListener` should not be released before the IM SDK is uninit
// otherwise the message callback cannot be called.
SimpleMsgListener simpleMsgListener;
V2TIMManager::GetInstance()->AddSimpleMsgListener(&simpleMsgListener);
```

Using advanced message listener

The receiver can receive a custom one-to-one or group message by using the advanced message listener in the following steps:

1. Call `addAdvancedMsgListener` to set the event listener.
2. Listen for the `onRecvNewMessage` callback (Java / Swift / Objective-C / C++) to receive custom messages.
3. To stop receiving messages, call `removeAdvancedMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

Java

Swift

Objective-C

C++

```
// Set the event listener
V2TIMManager.getMessageManager().addAdvancedMsgListener(v2TIMAdvancedMsgListener);
```

```
// Receive the message
public void onRecvNewMessage(V2TIMMessage msg) {
    // Parse the `groupID` and `userID`
    String groupID = msg.getGroupID();
    String userID = msg.getUserID();

    // Determine whether it's a one-to-one chat or group chat
    // If `groupID` is not empty, the message is from a group chat; if `userID` is

    // Parse the custom message in `msg`
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_CUSTOM) {
        V2TIMCustomElem customElem = msg.getCustomElem();
        String data = new String(customElem.getData());
        Log.i("onRecvNewMessage", "customData:" + data);
    }
}
```

```
// Set the event listener
V2TIMManager.shared.addAdvancedMsgListener(listener: self)

// Receive the message
func onRecvNewMessage(_ msg: V2TIMMessage) {
    // Parse the `groupID` and `userID`
    let groupID = msg.groupID
    let userID = msg.userID

// Determine whether it's a one-to-one chat or group chat
    // If `groupID` is not empty, the message is from a group chat; if `userID` is
    if !groupID.isEmpty {
        print("Received a group message in group: \\(groupID)")
    } else if !userID.isEmpty {
        print("Received a one-on-one message from user: \\(userID)")
    } else {
        print("Received a message with no identifiable sender.")
    }

    // Parse the custom message in `msg`
    if msg.elemType == .V2TIM_ELEM_TYPE_TEXT {
        if let textElem = msg.textElem {
            let text = textElem.text
            print("onRecvNewMessage, text: \\(text)")
        }
    } else {
        print("Received a non-text message.")
    }
```

```
}
```

```
// Set the event listener
[[V2TIMManager sharedInstance] addAdvancedMsgListener:self];

/// Receive the message
/// @param msg Message object
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    // Parse the `groupID` and `userID`
    NSString *groupID = msg.groupID;
    NSString *userID = msg.userID;

    // Determine whether it's a one-to-one chat or group chat
    // If `groupID` is not empty, the message is from a group chat; if `userID` is

    // Parse the custom message in `msg`
    if (msg.elemType == V2TIM_ELEM_TYPE_CUSTOM) {
        V2TIMCustomElem *customElem = msg.customElem;
        NSData *customData = customElem.data;
        NSLog(@"onRecvNewMessage, customData: %@", customData);
    }
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    /**
     * Receive the message
     *
     * @param message Message
     */
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        // Parse the `groupID` and `userID`
        V2TIMString groupID = message.groupID;
        V2TIMString userID = message.userID;

        // Determine whether it's a one-to-one chat or group chat
        // If `groupID` is not empty, the message is from a group chat;
        // if `userID` is not empty, the message is from a one-to-one chat.

        // Parse the custom message in `message`
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_CUSTOM) {
                auto customElem = static_cast<V2TIMCustomElem*>(elem);
                V2TIMBuffer data = customElem->data;
                // Parse the message and display it on the UI
```

```
                std::cout << "data:"
                          << std::string{reinterpret_cast<const char*>(data.Data())
                          << std::endl;
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Receiving a Rich Media Message

The receiver can receive a rich media message **only by using** the advanced message listener in the following steps:

1. Call the `addAdvancedMsgListener` API to set the advanced message listener.
2. Listen for the `onRecvNewMessage` callback (Java / Swift / Objective-C / C++) to receive the `V2TIMMessage` message.
3. Parse the `elemType` attribute in the `V2TIMMessage` message and then parse the message again according to the type to get the specific content of the elements in the message.
4. To stop receiving messages, call `removeAdvancedMsgListener` to remove the listener. This step is optional and can be performed as needed.

Image message

The image in an image message can be in three formats: original, large, and thumbnail. The latter two are automatically generated by the SDK during message sending and can be ignored.

Large image: A large image is an image obtained after the original image is proportionally compressed. After the compression, the height or width (whichever is shorter) is equal to 720 pixels.

Thumbnail: A thumbnail is an image obtained after the original image is proportionally compressed. After the compression, the height or width (whichever is shorter) is equal to 198 pixels.

After the image message is received, we recommend you call the SDK's `downloadImage` (Java / Swift / Objective-C / C++) to download the image and then render it to the UI layer.

To avoid repeated download and save resources, we recommend you set the `uuid` attribute value of the `V2TIMImage` object to the image download path to identify the image.

The following sample code demonstrates how to parse the image content from `V2TIMMessage` :

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_IMAGE) {
        // Image message
        V2TIMImageElem v2TIMImageElem = msg.getImageElem();
        // The image in an image message can be in three formats: original, large,
        // Large image: A large image is an image obtained after the original image
        // Thumbnail: A thumbnail is an image obtained after the original image is
        List<V2TIMImageElem.V2TIMImage> imageList = v2TIMImageElem.getImageList();
        for (V2TIMImageElem.V2TIMImage v2TIMImage : imageList) {
            // Image ID, which is an internal ID and can be used as an external cac
            String uuid = v2TIMImage.getUUID();
            // Image types, encompassing three distinct varieties: V2TIM_IMAGE_TYPE
            int imageType = v2TIMImage.getType();
            // Image size (bytes)
            int size = v2TIMImage.getSize();
            // Image width
            int width = v2TIMImage.getWidth();
            // Image height
            int height = v2TIMImage.getHeight();
            // Image download url
            String url = v2TIMImage.getUrl();
            // Set the image download path `imagePath`. Here, `uuid` can be used as
            String imagePath = "/sdcard/im/image/" + "myUserID" + uuid;
            File imageFile = new File(imagePath);
            // Determine whether there is a downloaded image file in `imagePath`
            if (!imageFile.exists()) {
                // Download the image
                v2TIMImage.downloadImage(imagePath, new V2TIMDownloadCallback() {
                    @Override
                    public void onProgress(V2TIMElem.V2ProgressInfo progressInfo) {
                        // Callback for download progress. `v2ProgressInfo.getCurre
                    }
                    @Override
                    public void onError(int code, String desc) {
                        // Download failed
                    }
                    @Override
                    public void onSuccess() {
                        // Downloaded
                    }
                });
            } else {
                // The image already exists.
            }
```

```
        }
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.elemType == .V2TIM_ELEM_TYPE_IMAGE {
        guard let imageElem = msg.imageElem else { return }
        // The image in an image message can be in three formats: original, large,
        // Large image: A large image is an image obtained after the original image
        // Thumbnail: A thumbnail is an image obtained after the original image is
        let imageList = imageElem.imageList
        for timImage in imageList {
            // Image ID, which is an internal ID and can be used as an external cac
            let uuid = timImage.uuid
            // Image type
            let type = timImage.type
            // Image size (bytes)
            let size = timImage.size
            let width = timImage.width
            let height = timImage.height
            // Image download url
            let url = timImage.url
            // Set the image download path `imagePath`. Here, `uuid` can be used as
            let imagePath = NSTemporaryDirectory().appending("testImage\\(timImage.

            // Determine whether there is a downloaded image file in `imagePath`
            if !FileManager.default.fileExists(atPath: imagePath) {
                // Download the image
                timImage.downloadImage(path: imagePath, progress: { curSize, totalS
                    print("Image download progress:curSize:\\(curSize), totalSize:
                }, succ: {

                    print("Image download success")
                }, fail: { code, msg in
                    // Download failed
                    print("failed:code:\\(code), msg: \\(msg)")
                })
            } else {
                // The image already exists.
                print(" The image already exists.:\\(imagePath)")
            }

            print("Image info:uuid: \\(uuid), type: \\(type.rawValue), size: \\(si
        }
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.elemType == V2TIM_ELEM_TYPE_IMAGE) {
        V2TIMImageElem *imageElem = msg.imageElem;
        // The list of original images, large images, and thumbnails
        NSArray<V2TIMImage *> *imageList = imageElem.imageList;
        for (V2TIMImage *timImage in imageList) {
            // Image ID, which is an internal ID and can be used as an external cac
            NSString *uuid = timImage.uuid;
            // Image type
            V2TIMImageType type = timImage.type;
            // Image size (bytes)
            int size = timImage.size;
            // Image width
            int width = timImage.width;
            // Image height
            int height = timImage.height;
            // Image download url
            NSString * url = timImage.url;
            // Set the image download path `imagePath`. Here, `uuid` can be used as
            NSString *imagePath = [NSTemporaryDirectory() stringByAppendingPathComp
            // Determine whether there is a downloaded image file in `imagePath`
            if (![[NSFileManager defaultManager] fileExistsAtPath:imagePath]) {
                // Download the image
                [timImage downloadImage:imagePath progress:^(NSInteger curSize, NSI
                    // Download progress
                    NSLog(@"Image download progress: curSize: %lu,totalSize:%lu",cu
                } succ:^{
                    // Downloaded successfully
                    NSLog(@"Image downloaded");
                } fail:^(int code, NSString *msg) {
                    // Download failed
                    NSLog(@"Failed to download the image: code: %d,msg:%@",code,msg
                }];
            } else {
                // The image already exists.
            }
            NSLog(@"Image information: uuid:%@, type:%ld, size:%d, width:%d, height
        }
    }
}
```

```
class DownloadCallback final : public V2TIMDownloadCallback {
public:
    using SuccessCallback = std::function<void()>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using DownLoadProgressCallback = std::function<void(uint64_t, uint64_t)>;
```

```
    DownloadCallback() = default;
    ~DownloadCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     DownLoadProgressCallback download_progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        download_progress_callback_ = std::move(download_progress_callback);
    }
    void OnSuccess() override {
        if (success_callback_) {
            success_callback_();
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnDownLoadProgress(uint64_t currentSize, uint64_t totalSize) override {
        if (download_progress_callback_) {
            download_progress_callback_(currentSize, totalSize);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    DownLoadProgressCallback download_progress_callback_;
};

class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_IMAGE) {
                // Image message
                auto imageElem = static_cast<V2TIMImageElem*>(elem);
                // The image in an image message can be in three formats: original,
                // The latter two are automatically generated by the SDK during mes
                // Large image: A large image is an image obtained after the origin
                // After the compression, the height or width (whichever is shorter
                // Thumbnail: A thumbnail is an image obtained after the original i
                // After the compression, the height or width (whichever is shorter
                V2TIMImageVector imageList = imageElem->imageList;
```

```
                for (size_t i = 0; i < imageList.Size(); ++i) {
                    V2TIMImage& image = imageList[i];
                    // Image ID, which is an internal ID and can be used as an exte
                    V2TIMString uuid = image.uuid;
                    // Image type
                    V2TIMImageType type = image.type;
                    // Image size (bytes) when type == V2TIMImageType::V2TIM_IMAGE_
                    uint64_t size = image.size;
                    // Image width
                    uint32_t width = image.width;
                    // Image height
                    uint32_t height = image.height;
                     // Image download url
                    V2TIMString url = image.url;
                    // Set the image download path `imagePath`.
                    // Here, `uuid` can be used as an identifier to avoid repeated
                    std::filesystem::path imagePath = u8"./File/Image/"s + uuid.CSt
                    // Determine whether there is a downloaded image file in `image
                    if (!std::filesystem::exists(imagePath)) {
                        std::filesystem::create_directories(imagePath.parent_path()
                        auto callback = new DownloadCallback{};
                        callback->SetCallback(
                            [=]() {
                                // Downloaded
                                delete callback;
                            },
                            [=](int error_code, const V2TIMString& error_message) {
                                // Download failed
                                delete callback;
                            },
                            [=](uint64_t currentSize, uint64_t totalSize) {
                                // Callback for download progress.
                                // `currentSize` indicates the downloaded file size
                                // and `totalSize` indicates the total file size.
                            });
                        image.DownloadImage(imagePath.string().c_str(), callback);
                    } else {
                        // The image already exists.
                    }
                }
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
```

```
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
                    V2TIMString url = image.url;
```

Video message

After a video message is received, a video preview is displayed on the chat page, and the video is played back after the user clicks the message.

Two steps are required:

1. Download the video screenshot. We recommend you call the SDK's `downloadSnapshot` (Java / Swift / Objective-C / C++) for download.

2. Download the video. We recommend you call the SDK's `downloadVideo` (Java / Swift / Objective-C / C++) for download.

To avoid repeated download and save resources, we recommend you set the `videoUUID` attribute value of the `V2TIMVideoElem` object to the video download path to identify the video.

The following sample code demonstrates how to parse the video content from `V2TIMMessage` :

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_VIDEO) {
        // Video message
        V2TIMVideoElem v2TIMVideoElem = msg.getVideoElem();
        // Video screenshot ID, which is an internal ID and can be used as an exter
        String snapshotUUID = v2TIMVideoElem.getSnapshotUUID();
        // Video screenshot file size
        int snapshotSize = v2TIMVideoElem.getSnapshotSize();
        // Video screenshot width
        int snapshotWidth = v2TIMVideoElem.getSnapshotWidth();
        // Video screenshot height
        int snapshotHeight = v2TIMVideoElem.getSnapshotHeight();
        // Video ID, which is an internal ID and can be used as an external cache k
        String videoUUID = v2TIMVideoElem.getVideoUUID();
        // Video file size
        int videoSize = v2TIMVideoElem.getVideoSize();
        // Video duration
        int duration = v2TIMVideoElem.getDuration();
        // Set the video screenshot file path. Here, `uuid` can be used as an ident
        String snapshotPath = "/sdcard/im/snapshot/" + "myUserID" + snapshotUUID;
        File snapshotFile = new File(snapshotPath);
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
        if (!snapshotFile.exists()) {
            v2TIMVideoElem.downloadSnapshot(snapshotPath, new V2TIMDownloadCallback
                @Override
                public void onProgress(V2TIMElem.V2ProgressInfo progressInfo) {
                    // Callback for download progress. `v2ProgressInfo.getCurrentSi
                }
                @Override
                public void onError(int code, String desc) {
                    // Download failed
                }
                @Override
                public void onSuccess() {
                    // Downloaded
                }
            });
        } else {
            // The file already exists.
        }

        // Set the video file path. Here, `uuid` can be used as an identifier to av
        String videoPath = "/sdcard/im/video/" + "myUserID" + videoUUID;
        File videoFile = new File(videoPath);
        if (!videoFile.exists()) {
            v2TIMVideoElem.downloadVideo(videoPath, new V2TIMDownloadCallback() {
                @Override
                public void onProgress(V2TIMElem.V2ProgressInfo progressInfo) {
                // Callback for download progress. `v2ProgressInfo.getCurrentSize()
                }
                @Override
                public void onError(int code, String desc) {
                // Download failed
                }
                @Override
                public void onSuccess() {
                // Downloaded
                }
            });
        } else {
            // The file already exists.
        }
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.elemType == .V2TIM_ELEM_TYPE_VIDEO {
        let videoElem = msg.videoElem
```

```
        // Video screenshot ID, which is an internal ID and can be used as an exter
        let snapshotUUID = videoElem?.snapshotUUID
        // Video screenshot file size
        let snapshotSize = videoElem?.snapshotSize

        let snapshotWidth = videoElem?.snapshotWidth

        let snapshotHeight = videoElem?.snapshotHeight
        // Video ID, which is an internal ID and can be used as an external cache k
        let videoUUID = videoElem?.videoUUID

        let videoSize = videoElem?.videoSize

        let duration = videoElem?.duration

        // Set the video screenshot file path. Here, `uuid` can be used as an ident
        let snapshotPath = NSTemporaryDirectory().appending("testVideoSnapshot\\(sn

        if !FileManager.default.fileExists(atPath: snapshotPath) {
            // Download the video screenshot
            videoElem?.downloadSnapshot(path: snapshotPath, progress: { curSize, to

                print("progress：curSize：\\(curSize), totalSize: \\(totalSize)")
            }, succ: {
                print("succ")
            }, fail: { code, msg in

                print("failed code：\\(code), msg: \\(msg)")
            })
        } else {
            // The video Snapshot already exists.
        }

        print("video info：snapshotUUID: \\(snapshotUUID), snapshotSize: \\(snapshot

        // Set the video file path. Here, `uuid` can be used as an identifier to av
        let videoPath = NSTemporaryDirectory().appending("testVideo\\(videoUUID)")

        if !FileManager.default.fileExists(atPath: videoPath) {
            // downloadVideo
            videoElem?.downloadVideo(path: videoPath, progress: { curSize, totalSiz

                print("downloadVideo progress curSize：\\(curSize), totalSize: \\(to
            }, succ: {
                print("downloadVideo success")
            }, fail: { code, msg in
```

```
                print("code：\\(code), msg: \\(msg)")
            })
        } else {
            // The video already exists.
        }
        print("video info：videoUUID: \\(videoUUID), videoSize: \\(videoSize), durat
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.elemType == V2TIM_ELEM_TYPE_VIDEO) {
        V2TIMVideoElem *videoElem = msg.videoElem;
        // Video screenshot ID, which is an internal ID and can be used as an exter
        NSString *snapshotUUID = videoElem.snapshotUUID;
        // Video screenshot file size
        int snapshotSize = videoElem.snapshotSize;
        // Video screenshot width
        int snapshotWidth = videoElem.snapshotWidth;
        // Video screenshot height
        int snapshotHeight = videoElem.snapshotHeight;
        // Video ID, which is an internal ID and can be used as an external cache k
        NSString *videoUUID = videoElem.videoUUID;
        // Video file size
        int videoSize = videoElem.videoSize;
        // Video duration
        int duration = videoElem.duration;
        // Set the video screenshot file path. Here, `uuid` can be used as an ident
        NSString *snapshotPath = [NSTemporaryDirectory() stringByAppendingPathCompo
        if (![[NSFileManager defaultManager] fileExistsAtPath:snapshotPath]) {
            // Download the video screenshot
            [videoElem downloadSnapshot:snapshotPath progress:^(NSInteger curSize,
                // Download progress
                NSLog(@"%@", [NSString stringWithFormat:@"Video screenshot download
            } succ:^{
                // Downloaded successfully
                NSLog(@"Video screenshot downloaded");
            } fail:^(int code, NSString *msg) {
                // Download failed
                NSLog(@"%@", [NSString stringWithFormat:@"Failed to download the vi
            }];
        } else {
            // The video screenshot already exists.
        }
        NSLog(@"Video screenshot information: snapshotUUID:%@, snapshotSize:%d, sna

        // Set the video file path. Here, `uuid` can be used as an identifier to av
```

```
        NSString *videoPath = [NSTemporaryDirectory() stringByAppendingPathComponen
        if (![[NSFileManager defaultManager] fileExistsAtPath:videoPath]) {
            // Download the video
            [videoElem downloadVideo:videoPath progress:^(NSInteger curSize, NSInte
                // Download progress
                NSLog(@"%@", [NSString stringWithFormat:@"Video download progress:
            } succ:^{
                // Downloaded successfully
                NSLog(@"Video downloaded");
            } fail:^(int code, NSString *msg) {
                // Download failed
                NSLog(@"%@", [NSString stringWithFormat:@"Failed to download the vi
            }];
        } else {
            // The video already exists.
        }
        NSLog(@"Video information: videoUUID:%@, videoSize:%d, duration:%d, videoPa
    }
}
```

```
class DownloadCallback final : public V2TIMDownloadCallback {
public:
    using SuccessCallback = std::function<void()>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using DownLoadProgressCallback = std::function<void(uint64_t, uint64_t)>;

    DownloadCallback() = default;
    ~DownloadCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     DownLoadProgressCallback download_progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        download_progress_callback_ = std::move(download_progress_callback);
    }
    void OnSuccess() override {
        if (success_callback_) {
            success_callback_();
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnDownLoadProgress(uint64_t currentSize, uint64_t totalSize) override {
```

```
        if (download_progress_callback_) {
            download_progress_callback_(currentSize, totalSize);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    DownLoadProgressCallback download_progress_callback_;
};

class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_VIDEO) {
                // Video message
                auto videoElem = static_cast<V2TIMVideoElem*>(elem);
                // Video ID, which is an internal ID and can be used as an external
                V2TIMString videoUUID = videoElem->videoUUID;
                // Video file size
                uint64_t videoSize = videoElem->videoSize;
                // Video type
                V2TIMString videoType = videoElem->videoType;
                // Video duration
                uint32_t duration = videoElem->duration;
                // Video screenshot ID, which is an internal ID and can be used as
                V2TIMString snapshotUUID = videoElem->snapshotUUID;
                // Video screenshot file size
                uint64_t snapshotSize = videoElem->snapshotSize;
                // Video screenshot width
                uint32_t snapshotWidth = videoElem->snapshotWidth;
                // Video screenshot height
                uint32_t snapshotHeight = videoElem->snapshotHeight;

                // Set the video file path. Here, `uuid` can be used as an identifi
                std::filesystem::path videoPath = u8"./File/Video/"s + videoUUID.CS
                if (!std::filesystem::exists(videoPath)) {
                    std::filesystem::create_directories(videoPath.parent_path());
                    auto callback = new DownloadCallback{};
                    callback->SetCallback(
                        [=]() {
                            // Downloaded
                            delete callback;
                        },
                        [=](int error_code, const V2TIMString& error_message) {
```

```
                            // Download failed
                            delete callback;
                        },
                        [=](uint64_t currentSize, uint64_t totalSize) {
                            // Callback for download progress.
                            // `currentSize` indicates the downloaded file size,
                            // and `totalSize` indicates the total file size.
                        });
                    videoElem->DownloadVideo(videoPath.string().c_str(), callback);
                } else {
                    // The file already exists.
                }

                // Set the video screenshot file path. Here, `uuid` can be used as
                std::filesystem::path snapshotPath = u8"./File/Snapshot/"s + snapsh
                if (!std::filesystem::exists(snapshotPath)) {
                    std::filesystem::create_directories(snapshotPath.parent_path())
                    auto callback = new DownloadCallback{};
                    callback->SetCallback(
                        [=]() {
                            // Downloaded
                            delete callback;
                        },
                        [=](int error_code, const V2TIMString& error_message) {
                            // Download failed
                            delete callback;
                        },
                        [=](uint64_t currentSize, uint64_t totalSize) {
                            // Callback for download progress.
                            // `currentSize` indicates the downloaded file size,
                            // and `totalSize` indicates the total file size.
                        });
                    videoElem->DownloadSnapshot(snapshotPath.string().c_str(), call
                } else {
                    // The file already exists.
                }
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Audio message

After the audio message is received, we recommend you call the SDK's `downloadSound` (Java / Swift / Objective-C / C++) to download the audio and then play it back.

To avoid repeated download and save resources, we recommend you set the `uuid` attribute value of the `V2TIMSoundElem` object to the audio download path to identify the audio.

The following sample code demonstrates how to parse the audio content from `V2TIMMessage` :

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_SOUND) {
        // Audio message
        V2TIMSoundElem v2TIMSoundElem = msg.getSoundElem();
        // Audio ID, which is an internal ID and can be used as an external cache k
        String uuid = v2TIMSoundElem.getUUID();
        // Audio file size
        int dataSize = v2TIMSoundElem.getDataSize();
        // Audio duration
        int duration = v2TIMSoundElem.getDuration();
        // Set the audio file path `soundPath`. Here, `uuid` can be used as an iden
        String soundPath = "/sdcard/im/sound/" + "myUserID" + uuid;
        File imageFile = new File(soundPath);
        // Determine whether there is a downloaded audio file in `soundPath`
        if (!imageFile.exists()) {
            v2TIMSoundElem.downloadSound(soundPath, new V2TIMDownloadCallback() {
                @Override
                public void onProgress(V2TIMElem.V2ProgressInfo progressInfo) {
                    // Callback for download progress. `v2ProgressInfo.getCurrentSi
                }
                @Override
                public void onError(int code, String desc) {
                    // Download failed
                }
                @Override
                public void onSuccess() {
                    // Downloaded
                }
            });
        } else {
            // The file already exists.
```

```
        }
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.elemType == .V2TIM_ELEM_TYPE_SOUND {
        guard let soundElem = msg.soundElem else { return }
        // Audio ID, which is an internal ID and can be used as an external cache k
        let uuid = soundElem.uuid
        // Audio file size
        let dataSize = soundElem.dataSize
        // Audio duration
        let duration = soundElem.duration
        // Set the audio file path `soundPath`. Here, `uuid` can be used as an iden
        let soundPath = NSTemporaryDirectory().appending("testSound\\(uuid)")

        // Determine whether there is a downloaded audio file in `soundPath`
        if !FileManager.default.fileExists(atPath: soundPath) {
            // downloadSound
            soundElem.downloadSound(path: soundPath, progress: { curSize, totalSize
                // Download progress
                print("Download progress：curSize：\\(curSize), totalSize: \\(totalS
            }, succ: {
                print("Download success")
            }, fail: { code, msg in
                print("error code：\\(code), msg: \\(msg)")
            })
        } else {
            // The audio already exists.
            print("The audio already exists：\\(soundPath)")
        }

        print("audio info：uuid: \\(uuid), dataSize: \\(dataSize), duration: \\(dura
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.elemType == V2TIM_ELEM_TYPE_SOUND) {
        V2TIMSoundElem *soundElem = msg.soundElem;
        // Audio ID, which is an internal ID and can be used as an external cache k
        NSString *uuid = soundElem.uuid;
        // Audio file size
        int dataSize = soundElem.dataSize;
        // Audio duration
        int duration = soundElem.duration;
        // Set the audio file path `soundPath`. Here, `uuid` can be used as an iden
```

```
        NSString *soundPath = [NSTemporaryDirectory() stringByAppendingPathComponen
        // Determine whether there is a downloaded audio file in `soundPath`
        if (![[NSFileManager defaultManager] fileExistsAtPath:soundPath]) {
            // Download the audio
            [soundElem downloadSound:soundPath progress:^(NSInteger curSize, NSInte
                // Download progress
                NSLog(@"Audio download progress: curSize: %lu,totalSize:%lu",curSiz
            } succ:^{
                // Downloaded successfully
                NSLog(@"Audio downloaded");
            } fail:^(int code, NSString *msg) {
                // Download failed
                NSLog(@"Failed to download the audio: code: %d,msg:%@",code,msg);
            }];
        } else {
            // The audio already exists.
        }
        NSLog(@"Audio information: uuid:%@, dataSize:%d, duration:%d, soundPath:%@"
    }
}
```

```
class DownloadCallback final : public V2TIMDownloadCallback {
public:
    using SuccessCallback = std::function<void()>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using DownLoadProgressCallback = std::function<void(uint64_t, uint64_t)>;

    DownloadCallback() = default;
    ~DownloadCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     DownLoadProgressCallback download_progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        download_progress_callback_ = std::move(download_progress_callback);
    }
    void OnSuccess() override {
        if (success_callback_) {
            success_callback_();
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
```

```
    void OnDownLoadProgress(uint64_t currentSize, uint64_t totalSize) override {
        if (download_progress_callback_) {
            download_progress_callback_(currentSize, totalSize);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    DownLoadProgressCallback download_progress_callback_;
};

class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_SOUND) {
                // Audio message
                auto soundElem = static_cast<V2TIMSoundElem*>(elem);
                // Audio ID, which is an internal ID and can be used as an external
                V2TIMString uuid = soundElem->uuid;
                // Audio file size
                uint64_t dataSize = soundElem->dataSize;
                // Audio duration (seconds)
                uint32_t duration = soundElem->duration;
                // Set the audio file path `soundPath`. Here, `uuid` can be used as
                std::filesystem::path soundPath = u8"./File/Sound/"s + uuid.CString
                // Determine whether there is a downloaded audio file in `soundPath
                if (!std::filesystem::exists(soundPath)) {
                    std::filesystem::create_directories(soundPath.parent_path());
                    auto callback = new DownloadCallback{};
                    callback->SetCallback(
                        [=]() {
                            // Downloaded
                            delete callback;
                        },
                        [=](int error_code, const V2TIMString& error_message) {
                            // Download failed
                            delete callback;
                        },
                        [=](uint64_t currentSize, uint64_t totalSize) {
                            // Callback for download progress.
                            // `currentSize` indicates the downloaded file size,
                            // and `totalSize` indicates the total file size.
                        });
                    soundElem->DownloadSound(soundPath.string().c_str(), callback);
```

```
                } else {
                    // The file already exists.
                }
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

File message

After the file message is received, we recommend you call the SDK's `downloadFile` (Java / Swift / Objective-C / C++) to download the file and then display it.

To avoid repeated download and save resources, we recommend you set the `uuid` attribute value of the `V2TIMFileElem` object to the file download path to identify the file.

The following sample code demonstrates how to parse the file content from `V2TIMMessage` :

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_FILE) {
        // File message
        V2TIMFileElem v2TIMFileElem = msg.getFileElem();
        // File ID, which is an internal ID and can be used as an external cache ke
        String uuid = v2TIMFileElem.getUUID();
        // Filename
        String fileName = v2TIMFileElem.getFileName();
        // File size
        int fileSize = v2TIMFileElem.getFileSize();
        // Set the file path. Here, `uuid` can be used as an identifier to avoid re
        String filePath = "/sdcard/im/file/" + "myUserID" + uuid;
        File file = new File(filePath);
        if (!file.exists()) {
            v2TIMFileElem.downloadFile(filePath, new V2TIMDownloadCallback() {
                @Override
                public void onProgress(V2TIMElem.V2ProgressInfo progressInfo) {
                    // Callback for download progress. `v2ProgressInfo.getCurrentSi
                }
```

```
                @Override
                public void onError(int code, String desc) {
                    // Download failed
                }
                @Override
                public void onSuccess() {
                    // Downloaded
                }
            });
        } else {
            // The file already exists.
        }
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.elemType == .V2TIM_ELEM_TYPE_FILE {
        guard let fileElem = msg.fileElem else { return }
        // File ID, which is an internal ID and can be used as an external cache ke
        let uuid = fileElem.uuid

        let filename = fileElem.filename

        let fileSize = fileElem.fileSize
        // Set the file path. Here, `uuid` can be used as an identifier to avoid re
        let filePath = NSTemporaryDirectory().appending("testFile\\(uuid)")

        if !FileManager.default.fileExists(atPath: filePath) {
            // downloadFile
            fileElem.downloadFile(path: filePath, progress: { curSize, totalSize in
                print("downloadFile progress：curSize：\\(curSize), totalSize: \\(to
            }, succ: {

                print("downloadFile success")
            }, fail: { code, msg in
                print("error code：\\(code), msg: \\(msg)")
            })
        } else {
            print("file exist \\(filePath)")
        }

        print("file info：uuid: \\(uuid), filename: \\(filename), fileSize: \\(file
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
```

```
    if (msg.elemType == V2TIM_ELEM_TYPE_FILE) {
        V2TIMFileElem *fileElem = msg.fileElem;
        // File ID, which is an internal ID and can be used as an external cache ke
        NSString *uuid = fileElem.uuid;
        // Filename
        NSString *filename = fileElem.filename;
        // File size
        int fileSize = fileElem.fileSize;
        // Set the file path. Here, `uuid` can be used as an identifier to avoid re
        NSString *filePath = [NSTemporaryDirectory() stringByAppendingPathComponent
        if (![[NSFileManager defaultManager] fileExistsAtPath:filePath]) {
            // Download the file
            [fileElem downloadFile:filePath progress:^(NSInteger curSize, NSInteger
                // Download progress
                NSLog(@"%@", [NSString stringWithFormat:@"File download progress: c
            } succ:^{
                // Downloaded successfully
                NSLog(@"File downloaded");
            } fail:^(int code, NSString *msg) {
                // Download failed
                NSLog(@"%@", [NSString stringWithFormat:@"Failed to download the fi
            }];
        } else {
            // The file already exists.
        }
        NSLog(@"File information: uuid:%@, filename:%@, fileSize:%d, filePath:%@",
    }
}
```

```
class DownloadCallback final : public V2TIMDownloadCallback {
public:
    using SuccessCallback = std::function<void()>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using DownLoadProgressCallback = std::function<void(uint64_t, uint64_t)>;

    DownloadCallback() = default;
    ~DownloadCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     DownLoadProgressCallback download_progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        download_progress_callback_ = std::move(download_progress_callback);
    }
    void OnSuccess() override {
        if (success_callback_) {
```

```
            success_callback_();
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnDownLoadProgress(uint64_t currentSize, uint64_t totalSize) override {
        if (download_progress_callback_) {
            download_progress_callback_(currentSize, totalSize);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    DownLoadProgressCallback download_progress_callback_;
};

class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_FILE) {
                // File message
                auto fileElem = static_cast<V2TIMFileElem*>(elem);
                // File ID, which is an internal ID and can be used as an external
                V2TIMString uuid = fileElem->uuid;
                // Filename
                V2TIMString filename = fileElem->filename;
                // File size
                uint64_t fileSize = fileElem->fileSize;
                // Set the file path. Here, `uuid` can be used as an identifier to
                std::filesystem::path filePath = u8"./File/File/"s + uuid.CString()
                if (!std::filesystem::exists(filePath)) {
                    std::filesystem::create_directories(filePath.parent_path());
                    auto callback = new DownloadCallback{};
                    callback->SetCallback(
                        [=]() {
                            // Downloaded
                            delete callback;
                        },
                        [=](int error_code, const V2TIMString& error_message) {
                            // Download failed
                            delete callback;
```

```
                        },
                        [=](uint64_t currentSize, uint64_t totalSize) {
                            // Callback for download progress.
                            // `currentSize` indicates the downloaded file size,
                            // and `totalSize` indicates the total file size.
                        });
                    fileElem->DownloadFile(filePath.string().c_str(), callback);
                } else {
                    // The file already exists.
                }
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Location message

After receiving the location message, the receiver can parse the latitude and longitude information directly from `V2TIMLocationElem`.

The following sample code demonstrates how to parse the location content from `V2TIMMessage`:

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_LOCATION) {
        // Geographical location message
        V2TIMLocationElem v2TIMLocationElem = msg.getLocationElem();
        // Geographical location information description
        String desc = v2TIMLocationElem.getDesc();
        // Longitude
        double longitude = v2TIMLocationElem.getLongitude();
        // Latitude
        double latitude = v2TIMLocationElem.getLatitude();
    }
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.elemType == .V2TIM_ELEM_TYPE_LOCATION {
        let locationElem = msg.locationElem
        // Geographical location information description
        let desc = locationElem?.desc
        // Longitude
        let longitude = locationElem?.longitude
        // Latitude
        let latitude = locationElem?.latitude
        print("location info：desc：\\(desc), longitude:\\(longitude), latitude:\\(l
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.elemType == V2TIM_ELEM_TYPE_LOCATION) {
        V2TIMLocationElem *locationElem = msg.locationElem;
        // Geographical location information description
        NSString *desc = locationElem.desc;
        // Longitude
        double longitude = locationElem.longitude;
        // Latitude
        double latitude = locationElem.latitude;
        NSLog(@"Geographical location information: desc: %@, longitude:%f, latitude
    }
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_LOCATION) {
                // Geographical location message
                auto locationElem = static_cast<V2TIMLocationElem*>(elem);
                // Geographical location information description
                V2TIMString desc = locationElem->desc;
                // Longitude
                double longitude = locationElem->longitude;
                // Latitude
                double latitude = locationElem->latitude;
            }
        }
    }
    // Other members ...
};
```

```
// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Emoji message

The SDK provides a passthrough channel only for emoji messages. For message content fields, see the definition of `V2TIMFaceElem` (Java / Swift / Objective-C / C++). Here, `index` and `data` can be customized.

For example, the sender can set `index` to `1` and `data` to `x12345` to indicate the smile emoji.

The receiver parses the received emoji message as `1` and `x12345` and displays the message as the smile emoji according to the preset rules.

The following sample code demonstrates how to parse the emoji content from `V2TIMMessage`:

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_FACE) {
        // Emoji message
        V2TIMFaceElem v2TIMFaceElem = msg.getFaceElem();
        // Emoji location
        int index = v2TIMFaceElem.getIndex();
        // Custom emoji data
        byte[] data = v2TIMFaceElem.getData();
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    guard msg.elemType == .V2TIM_ELEM_TYPE_FACE, let faceElem = msg.faceElem else {
        return
    }
    // Emoji location
    let index = faceElem.index
    // Custom emoji data
    guard let data = faceElem.data else {
        print("Emoji empty")
        return
    }

    print("Emoji information index: \\(index), data: \\(data)")
```

```
}

- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.elemType == V2TIM_ELEM_TYPE_FACE) {
        V2TIMFaceElem *faceElem = msg.faceElem;
        // Emoji location
        int index = faceElem.index;
        // Custom emoji data
        NSData *data = faceElem.data;
        NSLog(@"Emoji information: index: %d, data: %@", index, data);
    }
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_FACE) {
                // Emoji message
                auto faceElem = static_cast<V2TIMFaceElem*>(elem);
                // Emoji location
                uint32_t index = faceElem->index;
                // Custom emoji data
                V2TIMBuffer data = faceElem->data;
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Receiving a Broadcast Message

You can only use the Broadcast Message of Audio-Video Group to send broadcast messages to all AVChatRooms under the same AppID, and all online group members in the AVChatRooms can receive this message. The SDK cannot send broadcast messages, it can only receive messages from the REST API. The receiving of messages is still

©2013-2025 Tencent Cloud International Pte. Ltd.

in the `onRecvNewMessage` notification mentioned earlier. After receiving a message, you can determine whether the message is a broadcast message based on the `isBroadcastMessage` property.

Note：

1. To use this feature, you need to purchase the Pro edition、Pro Plus edition、Enterprise edition.
2. This feature is available only in SDK enhanced edition v6.5.2803 or later.
3. Only AVChatRoom supports broadcast message, other types of group are temporarily not supported.

Java

Swift

Objective-C

C++

```
public void onRecvNewMessage(V2TIMMessage msg) {
    if (msg.isBroadcastMessage) {
        // Received the broadcast message
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.isBroadcastMessage {
        // Received the broadcast message
     }
 }
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.isBroadcastMessage) {
        // Received the broadcast message
    }
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.isBroadcastMessage) {
            // Received the broadcast message
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Receiving a Message Containing Multiple Elements

1. Parse the first element object through the Message object.
2. Get the next element object through the `nextElem` method of the first element object. If the next object exists, an element object instance is returned; otherwise, `nil` or `null` is returned.

Sample code:

Java

Swift

Objective-C

C++

```
@Override
public void onRecvNewMessage(V2TIMMessage msg) {
    // View the first element
    int elemType = msg.getElemType();
    if (elemType == V2TIMMessage.V2TIM_ELEM_TYPE_TEXT) {
        // Text message
        V2TIMTextElem v2TIMTextElem = msg.getTextElem();
        String text = v2TIMTextElem.getText();
        // Check whether `v2TIMTextElem` is followed by more elements
        V2TIMElem elem = v2TIMTextElem.getNextElem();
        while (elem != null) {
            // Identify the element type. Here, `V2TIMCustomElem` is used as an exa
            if (elem instanceof V2TIMCustomElem) {
            V2TIMCustomElem customElem = (V2TIMCustomElem) elem;
            byte[] data = customElem.getData();
        }
        // Further check whether the current element is followed by more elements
        elem = elem.getNextElem();
        }
        // If the element is `null`, all the elements have been parsed.
    }
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    // View the first element
    if msg.elemType == .V2TIM_ELEM_TYPE_TEXT {
        let textElem = msg.textElem
        let text = textElem?.text
        print("text : \\(text)")
        var elem: V2TIMElem? = textElem?.getNextElem()
```

```
        while elem != nil {
            // Identify the element type
            if let customElem = elem as? V2TIMCustomElem {
                let customData = customElem.data
                print("Custom information : \\(customData)")
            }
            // Further check whether the current element is followed by more elemen
            elem = elem?.getNextElem()
        }
        // If the element is `nil`, all the elements have been parsed.
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    // View the first element
    if (msg.elemType == V2TIM_ELEM_TYPE_TEXT) {
        V2TIMTextElem *textElem = msg.textElem;
        NSString *text = textElem.text;
        NSLog(@"Text information: %@", text);
        // Check whether `textElem` is followed by more elements
        V2TIMElem *elem = textElem.nextElem;
        while (elem != nil) {
            // Identify the element type
            if ([elem isKindOfClass:[V2TIMCustomElem class]]) {
                V2TIMCustomElem *customElem = (V2TIMCustomElem *)elem;
                NSData *customData = customElem.data;
                NSLog(@"Custom information: %@",customData);
            }
            // Further check whether the current element is followed by more elemen
            elem = elem.nextElem;
        }
        // If the element is `nil`, all the elements have been parsed.
    }
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        // View the first element
        for (size_t i = 0; i < message.elemList.Size(); ++i) {
            V2TIMElem* elem = message.elemList[i];
            switch (elem->elemType) {
                case V2TIMElemType::V2TIM_ELEM_TYPE_NONE: {
                    // Unknown message
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_TEXT: {
```

```
                    // Text message
                    auto textElem = static_cast<V2TIMTextElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_CUSTOM: {
                    // Custom message
                    auto customElem = static_cast<V2TIMCustomElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_IMAGE: {
                    // Image message
                    auto imageElem = static_cast<V2TIMImageElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_SOUND: {
                    // Audio message
                    auto soundElem = static_cast<V2TIMSoundElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_VIDEO: {
                    // Video message
                    auto videoElem = static_cast<V2TIMVideoElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_FILE: {
                    // File message
                    auto fileElem = static_cast<V2TIMFileElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_LOCATION: {
                    // Geographical location message
                    auto locationElem = static_cast<V2TIMLocationElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_FACE: {
                    // Emoji message
                    auto faceElem = static_cast<V2TIMFaceElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_GROUP_TIPS: {
                    // Group @ message
                    auto mergerElem = static_cast<V2TIMMergerElem*>(elem);
                } break;
                case V2TIMElemType::V2TIM_ELEM_TYPE_MERGER: {
                    // Merged message
                    auto groupTipsElem = static_cast<V2TIMGroupTipsElem*>(elem);
                } break;
                default: {
                } break;
            }
        }
    }
    // Other members ...
};
```

```
// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

JavaScript

Last updated：2025-01-07 15:20:40

Feature Overview

To receive messages, you need to listen for the `MESSAGE_RECEIVED` event.

Listening for an Event

Note:

Call this API to listen for events before calling the `login` API in order to avoid missing the events delivered by the SDK.

API

```
chat.on(eventName, handler, context);
```

Parameters

Name	Type	Description
eventName	String	Event name. All event names are stored in the `TencentCloudChat.EVENT` variable. To view all the events, use `console.log(TencentCloudChat.EVENT)` .
handler	Function	Event handling method. When an event is triggered, this handler will be called to handle it.
context	* | undefined	The context expected for handler execution

Examples

```
let onMessageReceived = function(event) {
  // event.data - An array that stores `Message` objects - [Message]
  // For details of Message, please refer to https://www.tencentcloud.com/document/
  const messageList = event.data;
  messageList.forEach((message) => {
    if (message.type === TencentCloudChat.TYPES.MSG_TEXT) {
      // Text message
    } else if (message.type === TencentCloudChat.TYPES.MSG_IMAGE) {
      // Image message
    } else if (message.type === TencentCloudChat.TYPES.MSG_AUDIO) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      // Audio message
    } else if (message.type === TencentCloudChat.TYPES.MSG_VIDEO) {
      // Video message
    } else if (message.type === TencentCloudChat.TYPES.MSG_FILE) {
      // File message
    } else if (message.type === TencentCloudChat.TYPES.MSG_CUSTOM) {
      // Custom message
    } else if (message.type === TencentCloudChat.TYPES.MSG_MERGER) {
      // Merged message
    } else if (message.type === TencentCloudChat.TYPES.MSG_LOCATION) {
      // Geographical location message
    } else if (message.type === TencentCloudChat.TYPES.MSG_GRP_TIP) {
      // Group tip message
    } else if (message.type === TencentCloudChat.TYPES.MSG_GRP_SYS_NOTICE) {
      // Group system notification
    }
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

Canceling Event Listening

API

```
chat.off(eventName, handler, context);
```

Parameters

Name	Type	Description
eventName	String	Event name. All event names are stored in the `TencentCloudChat.EVENT` variable. To view all the events, use `console.log(TencentCloudChat.EVENT)` .
handler	Function	Event handling method. When an event is triggered, this handler will be called to handle it.
context	* | undefined	The context expected for handler execution

Examples

```
let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array that stores `Message` objects - [Message]
```

```
};
chat.off(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

Flutter

Last updated：2025-04-16 14:40:39

Overview

All types of messages (text, custom, and rich media messages) are listened for and received with the `addAdvancedMsgListener` API, with the callbacks defined in the protocol of `V2TimAdvancedMsgListener`.

Setting Message Listeners

Advanced message listener

Adding a listener

The receiver calls the `addAdvancedMsgListener` API (details) to add the advanced message listener. It is recommended to call the API early (such as after the chat panel is initialized) to ensure timely message receiving in the application.
Sample code:

```
// Create a message listener
V2TimAdvancedMsgListener listener = V2TimAdvancedMsgListener(
  onRecvC2CReadReceipt: (List<V2TimMessageReceipt> receiptList) {
    // Callback after marking a one-to-one message as read
  },
  onRecvMessageModified: (V2TimMessage message) {
    // `msg` is the modified message object
  },
  onRecvMessageReadReceipts: (List<V2TimMessageReceipt> receiptList) {
    // Callback after marking a group message as read
    receiptList.forEach((element) {
      element.groupID;  // Group ID
      element.msgID; // Message ID
      element.readCount;// Latest group message read count
      element.unreadCount;// Latest group message unread count
      element.userID; //  ID of the other party of the one-to-one message
    });
  },
  onRecvMessageRevoked: (String messageid) {
    // Process the recalled messages among locally maintained messages
  },
```

```
      onRecvNewMessage: (V2TimMessage message) async {
        // Process a text message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_TEXT) {
          message.textElem?.text;
        }
        // Use a custom message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_CUSTOM) {
          message.customElem?.data;
          message.customElem?.desc;
          message.customElem?.extension;
        }
        // Use an image message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_IMAGE) {
          message.imageElem
              ?.path; // Image upload path. This field applies only to the message
          message.imageElem?.imageList?.forEach((element) {
            // Traverse large images, original images, and thumbnails
            // Parse image attributes
            element?.height;
            element?.localUrl;
            element?.size;
            element?.type; // Large image, original image, or thumbnail
            element?.url;
            element?.uuid;
            element?.width;
          });
        }
        // Process a video message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_VIDEO) {
          // Parse the video message attributes such as thumbnail, playback address
          message.videoElem?.UUID;
          message.videoElem?.duration;
          message.videoElem?.localSnapshotUrl;
          message.videoElem?.localVideoUrl;
          message.videoElem?.snapshotHeight;
          message.videoElem?.snapshotPath;
          // ...
        }
        // Process an audio message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_SOUND) {
          // Parse the playback address, local address, size, and duration of the a
          message.soundElem?.UUID;
          message.soundElem?.dataSize;
          message.soundElem?.duration;
          message.soundElem?.localUrl;
          message.soundElem?.url;
          // ...
```

```
        }
        // Process a file message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_FILE) {
          // Parse the filename, size, URL and other info of the file message
          message.fileElem?.UUID;
          message.fileElem?.fileName;
          message.fileElem?.fileSize;
          message.fileElem?.localUrl;
          message.fileElem?.path;
          message.fileElem?.url;
        }
        // Process a location message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_LOCATION) {
          // Parse the geographical location information such as latitude, longitud
          message.locationElem?.desc;
          message.locationElem?.latitude;
          message.locationElem?.longitude;
        }
        // Process an emoji message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_FACE) {
          message.faceElem?.data;
          message.faceElem?.index;
        }
        // Process a group tip text message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_GROUP_TIPS) {
          message.groupTipsElem?.groupID; // Group
          message.groupTipsElem?.type; // Group tip type
          message.groupTipsElem?.opMember; // Operator profile
          message.groupTipsElem?.memberList; // User profile being operated
          message.groupTipsElem?.groupChangeInfoList; // Group information change d
          message.groupTipsElem?.memberChangeInfoList; // Group member change infor
          message.groupTipsElem?.memberCount; // Number of current online group mem
        }
        // Process a combined message
        if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_MERGER) {
          message.mergerElem?.abstractList;
          message.mergerElem?.isLayersOverLimit;
          message.mergerElem?.title;
          V2TimValueCallback<List<V2TimMessage>> download =
              await TencentImSDKPlugin.v2TIMManager
                  .getMessageManager()
                  .downloadMergerMessage(
                    msgID: message.msgID!,
                  );
          if (download.code == 0) {
            List<V2TimMessage>? messageList = download.data;
          }
```

```
        }
        if (message.textElem?.nextElem != null) {
          // Get the next element object through the `nextElem` method of the first
        }
      },
      onSendMessageProgress: (V2TimMessage message, int progress) {
        // File upload progress callback
      },
    );
    // Add an event listener for advanced messages
    TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .addAdvancedMsgListener(listener: listener);
```

Removing a listener

To stop receiving messages, the receiver calls `removeAdvancedMsgListener` API (details) to remove the advanced message listener.

Sample code:

```
// Create a listener
V2TimAdvancedMsgListener listener = V2TimAdvancedMsgListener(
    onRecvC2CReadReceipt: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageModified: (V2TimMessage message) {},
    onRecvMessageReadReceipts: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageRevoked: (String messageid) {},
    onRecvNewMessage: (V2TimMessage message) {},
    onSendMessageProgress: (V2TimMessage message, int progress) {},
  );
// Register the listener
  TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .addAdvancedMsgListener(listener: listener);
// Remove the listener
TencentImSDKPlugin.v2TIMManager.getMessageManager().removeAdvancedMsgListener(liste
```

Receiving Text Messages

The receiver can receive a one-to-one or group text message with the advanced message listener in the following steps:

1. Call `addAdvancedMsgListener` to set the event listener.
2. Listen for the `onRecvNewMessage` callback (details) that contains the custom message.

3. To stop receiving messages, call `removeAdvancedMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

```
V2TimAdvancedMsgListener listener = V2TimAdvancedMsgListener(
    onRecvC2CReadReceipt: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageModified: (V2TimMessage message) {},
    onRecvMessageReadReceipts: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageRevoked: (String messageid) {},
    onRecvNewMessage: (V2TimMessage message) {
      // Process a text message
      if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_TEXT){
        String text = message.textElem.text;
      }
    },
    onSendMessageProgress: (V2TimMessage message, int progress) {},
  );
  TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .addAdvancedMsgListener(listener: listener);
```

Receiving Custom Messages

The receiver can receive a custom one-to-one or group message with the advanced message listener in the following steps:

1. Call `addAdvancedMsgListener` to set the event listener.
2. Listen for the `onRecvNewMessage` callback (details) that contains the custom message.
3. To stop receiving messages, call `removeAdvancedMsgListener` to remove the listener. This step is optional and can be performed as needed.

Sample code:

```
V2TimAdvancedMsgListener listener = V2TimAdvancedMsgListener(
    onRecvC2CReadReceipt: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageModified: (V2TimMessage message) {},
    onRecvMessageReadReceipts: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageRevoked: (String messageid) {},
    onRecvNewMessage: (V2TimMessage message) {
      // Use a custom message
      if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_CUSTOM){
        String data = message.customElem.data;
        String desc = message.customElem.desc;
        String ext = message.customElem.extension;
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      }
    },
    onSendMessageProgress: (V2TimMessage message, int progress) {},
  );
  TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .addAdvancedMsgListener(listener: listener);
```

Receiving Rich Media Messages

The receiver can receive a rich media message **only with** the advanced message listener in the following steps:
1. Call the `addAdvancedMsgListener` API to set the advanced message listener.
2. Listen for the `onRecvNewMessage` callback (details) to get the `V2TimMessage` message.
3. Parse the `elemType` attribute in the `V2TIMMessage` message, and then parse the message again according to the type to get the specific content of the elements in the message.
4. To stop receiving messages, call `removeAdvancedMsgListener` to remove the listener. This step is optional and can be performed as needed.

Image message

The image in an image message can be in three formats: original, large, and thumbnail. The latter two are automatically generated by the SDK during message sending and can be ignored.
`Large` (large image): A large image is an image obtained after the original image is proportionally compressed. After the compression, the height or width (whichever is shorter) is equal to 720 pixels.
`Thumb` (thumbnail): A thumbnail is an image obtained after the original image is proportionally compressed. After the compression, the height or width (whichever is shorter) is equal to 198 pixels.
In the SDK for Flutter 5.0.4 or later, multimedia message online URLs (`url`) will not be returned when users get the message list. They need to be obtained by calling the `getMessageOnlineUrl` API (details).
In the SDK for Flutter SDK 5.0.4 or later, multimedia message local URLs (`localurl`) will no longer be returned by default. They will be returned only after you download messages successfully by calling the `downloadMessage` API (details).
The following sample code demonstrates how to parse the image content from `V2TimMessage` :

```
V2TimAdvancedMsgListener listener = V2TimAdvancedMsgListener(
    onRecvC2CReadReceipt: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageModified: (V2TimMessage message) {},
    onRecvMessageReadReceipts: (List<V2TimMessageReceipt> receiptList) {},
    onRecvMessageRevoked: (String messageid) {},
    onRecvNewMessage: (V2TimMessage message) async {
      // Use an image message
```

```
      if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_IMAGE) {
        String path = message
            .imageElem.path; // Image upload path. This field applies only to the m
        for (var item in message.imageElem.imageList) { // Traverse large images, o
          // Parse image attributes
          int height = item.height;
          String localUrl = item.localUrl;
          int size = item.size;
          int type = item.type; // Large image, thumbnail, or original image
          String url = item.url;
          String uuid = item.uuid;
          int width = item.width;
        }
      }
      // Get the multimedia message URL
      V2TimValueCallback<V2TimMessageOnlineUrl> getMessageOnlineUrlRes =
          await TencentImSDKPlugin.v2TIMManager
              .getMessageManager()
              .getMessageOnlineUrl(
                msgID: '', // Message ID
              );
      if (getMessageOnlineUrlRes.code == 0) {
        // Obtained successfully
      }

      // Download a multimedia message
      V2TimCallback downloadMessageRes = await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .downloadMessage(
              msgID: '', // Message ID
              messageType: 3, // Message type
              imageType: 0, // Image type. This field is valid only for image messa
              isSnapshot: false // Whether it is a video thumbnail. This field is v
              );
      if (downloadMessageRes.code == 0) {
        // Download successful
      }
    },
    onSendMessageProgress: (V2TimMessage message, int progress) {},
  );
  TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .addAdvancedMsgListener(listener: listener);
```

Video message

After a video message is received, a video preview is displayed on the chat page, and the video is played back after the user clicks the message.
The SDK for Flutter will download video messages when free, which can be directly used by you. Local videos and video thumbnails will be cleared when the application is uninstalled. The following sample code demonstrates how to parse video content from `V2TimMessage`:
In the SDK for Flutter 5.0.4 or later, multimedia message online URLs (`url`) will not be returned when users get the message list. They need to be obtained by calling the `getMessageOnlineUrl` API (details).
In the SDK for Flutter SDK 5.0.4 or later, multimedia message local URLs (`localurl`) will no longer be returned by default. They will be returned only after you download messages successfully by calling the `downloadMessage` API (details).

```
if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_VIDEO){
        // Parse the video message attributes such as thumbnail, playback add
  message.videoElem.UUID;
  message.videoElem.duration;
  message.videoElem.localSnapshotUrl;
  message.videoElem.localVideoUrl;
  message.videoElem.snapshotHeight;
  message.videoElem.snapshotPath;
  // ...
}

// Get the multimedia message URL
V2TimValueCallback<V2TimMessageOnlineUrl> getMessageOnlineUrlRes =
    await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .getMessageOnlineUrl(
          msgID: '', // Message ID
        );
if (getMessageOnlineUrlRes.code == 0) {
  // Obtained successfully
}

// Download a multimedia message
V2TimCallback downloadMessageRes = await TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .downloadMessage(
        msgID: '', // Message ID
        messageType: 5, // Message type
        imageType: 0, // Image type. This field is valid only for image messa
        isSnapshot: false // Whether it is a video thumbnail. This field is v
        );
if (downloadMessageRes.code == 0) {
  // Download successful
```

```
    }
```

Audio message

The SDK for Flutter will download audio messages when free, which can be directly used by you and will be cleared when the application is uninstalled. The following sample code demonstrates how to parse audio content from `V2TimMessage`:

In the SDK for Flutter 5.0.4 or later, multimedia message online URLs (`url`) will not be returned when users get the message list. They need to be obtained by calling the `getMessageOnlineUrl` API (details).

In the SDK for Flutter SDK 5.0.4 or later, multimedia message local URLs (`localurl`) will no longer be returned by default. They will be returned only after you download messages successfully by calling the `downloadMessage` API (details).

```
if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_SOUND){
          // Parse the playback address, local address, size, and duration of t
    message.soundElem.UUID;
    message.soundElem.dataSize;
    message.soundElem.duration;
    message.soundElem.localUrl;
    message.soundElem.url;
    // ...
    // Get the multimedia message URL
    V2TimValueCallback<V2TimMessageOnlineUrl> getMessageOnlineUrlRes =
        await TencentImSDKPlugin.v2TIMManager
            .getMessageManager()
            .getMessageOnlineUrl(
              msgID: '', // Message ID
            );
    if (getMessageOnlineUrlRes.code == 0) {
      // Obtained successfully
    }

    // Download a multimedia message
    V2TimCallback downloadMessageRes = await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .downloadMessage(
            msgID: '', // Message ID
            messageType: 4, // Message type
            imageType: 0, // Image type. This field is valid only for image mes
            isSnapshot: false // Whether it is a video thumbnail. This field is
            );
    if (downloadMessageRes.code == 0) {
      // Download successful
    }
}
```

File message

The SDK for Flutter will download file messages when free, which can be directly used by you and will be cleared when the application is uninstalled. The following sample code demonstrates how to parse file message content from `V2TimMessage`:

In the SDK for Flutter 5.0.4 or later, multimedia message online URLs (`url`) will not be returned when users get the message list. They need to be obtained by calling the `getMessageOnlineUrl` API (details).

In the SDK for Flutter SDK 5.0.4 or later, multimedia message local URLs (`localurl`) will no longer be returned by default. They will be returned only after you download messages successfully by calling the `downloadMessage` API (details).

```
if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_FILE){
          // Parse the filename, size, URL and other info of the file message
    message.fileElem.UUID;
    message.fileElem.fileName;
    message.fileElem.fileSize;
    message.fileElem.localUrl;
    message.fileElem.path;
    message.fileElem.url;
    // Get the multimedia message URL
    V2TimValueCallback<V2TimMessageOnlineUrl> getMessageOnlineUrlRes =
        await TencentImSDKPlugin.v2TIMManager
            .getMessageManager()
            .getMessageOnlineUrl(
              msgID: '', // Message ID
            );
    if (getMessageOnlineUrlRes.code == 0) {
      // Obtained successfully
    }

    // Download a multimedia message
    V2TimCallback downloadMessageRes = await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .downloadMessage(
            msgID: '', // Message ID
            messageType: 6, // Message type
            imageType: 0, // Image type. This field is valid only for image mes
            isSnapshot: false // Whether it is a video thumbnail. This field is
            );
    if (downloadMessageRes.code == 0) {
      // Download successful
    }
}
```

Geographical location message

After receiving the geographical location message, the receiver can parse the latitude and longitude information directly from `locationElem` .

The following sample code demonstrates how to parse the geographical location content from `V2TimMessage` :

```
if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_LOCATION){
                // Parse the geographical location information such as latitude, long
        message.locationElem.desc;
        message.locationElem.latitude;
        message.locationElem.longitude;
}
```

Emoji message

The SDK only provides a message passthrough channel for an emoji message. For message content fields, see `faceElem` (details), where the contents of `index` and `data` are customized by the user.

For example, the sender can set `index` to `1` and `data` to `x12345` to indicate the smile emoji.

The receiver parses the received emoji message as `1` and `x12345` and displays the message as the smile emoji according to the preset rules.

The following sample code demonstrates how to parse the emoji content from `V2TIMMessage` :

```
if (message.elemType == MessageElemType.V2TIM_ELEM_TYPE_FACE) {
        message.faceElem.data;
        message.faceElem.index;
}
```

Group tip message

Group tip messages are tips received by users other than ordinary messages in a group chat, such as "The admin removed alice from the group chat" and "bob renamed the group "xxxx"".

Note:

Group tip messages apply to group members only.

There are many types of group tip messages. For more information, see the definition of `V2TIMGroupTipsType` (details).

After receiving a group tip message, the receiver generally needs to:

1. Parse all fields in `V2TIMGroupTipsElem` .
2. Identify the message type according to `type` .
3. Merge other fields into the content for display according to the type.

©2013-2025 Tencent Cloud International Pte. Ltd.

Example:

`V2TIM_GROUP_TIPS_TYPE_GROUP_INFO_CHANGE` is parsed from the `type`, indicating that this is a notification of group profile change.

The receiver can get the operator information from `opMember` and the new group name from `groupChangeInfoList`.

At this point, the receiver can merge the "operator" and "new group name" to create a group tip, such as "Alice renamed the group as "group123"".

The following sample code demonstrates how to parse the tip content from `V2TIMMessage`:

```
if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_GROUP_TIPS){
        message.groupTipsElem.groupID; // Group
        message.groupTipsElem.type; // Group tip type
        message.groupTipsElem.opMember; // Operator profile
        message.groupTipsElem.memberList; // User profile being operated
        message.groupTipsElem.groupChangeInfoList; // Group information change deta
        message.groupTipsElem.memberChangeInfoList; // Group member change informat
        message.groupTipsElem.memberCount; // Number of current online group member
}
```

Receiving Messages Containing Multiple Element Objects

1. Parse the first element object from the `Message` object.

2. Get the next element object through the `nextElem` method of the first element object. If the next object exists, an element object instance is returned; otherwise, `null` is returned.

Sample code:

```
if(message.textElem.nextElem!=null){
   // There is a next message
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Unity

Last updated：2025-03-25 09:56:12

Feature Description

The `RecvNewMsgCallback` is used to listen for and receive all types of messages (text, custom, and rich media).

Setting a Message Listener

Adding a listener

The receiver calls the `AddRecvNewMsgCallback` (Details) to add a message listener. We recommend it be called early, such as after the chat page is initialized, to ensure timely message receiving in the application.

Removing a listener

To stop receiving messages, the receiver can call `RemoveRecvNewMsgCallback` (Details) to remove the message listener.

Receiving a Text Message

The receiver can receive a one-to-one or group text message by using a message listener in the following steps:
1. Call `AddRecvNewMsgCallback` to set the event listener.
2. Listen for the `RecvNewMsgCallback` callback to receive text messages.
3. To stop receiving messages, call `RemoveRecvNewMsgCallback` to remove the listener. This step is optional and can be performed as needed.
Sample code:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Text)
      {
         string text = elem.text_elem_content;
      }
    }
```

```
    }
})
```

Receiving a Custom Message

The receiver can receive a custom one-to-one or group message by using a message listener in the following steps:

1. Call `AddRecvNewMsgCallback` to set the event listener.
2. Listen for the `RecvNewMsgCallback` callback to receive custom messages.
3. To stop receiving messages, call `RemoveRecvNewMsgCallback` to remove the listener. This step is optional and can be performed as needed.

Sample code:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Custom)
      {
         string data = elem.custom_elem_data;
         string desc = elem.custom_elem_desc;
         string ext = elem.custom_elem_ext;
      }
    }
  }
})
```

Receiving a Rich Media Message

The receiver can receive a rich media message by using a message listener in the following steps:

1. Call `AddRecvNewMsgCallback` to set the event listener.
2. Listen for the `RecvNewMsgCallback` callback to receive rich media messages.
3. Parse the `elem_type` attribute in the `Message` message and then parse the message again according to the type to get the specific content of the elements in the message.
4. To stop receiving messages, call `RemoveRecvNewMsgCallback` to remove the listener. This step is optional and can be performed as needed.

Image message

The image in an image message can be in three formats: original, large, and thumbnail. The latter two are automatically generated by the SDK during message sending and can be ignored.
Large image: A large image is an image obtained after the original image is proportionally compressed. After the compression, the height or width (whichever is shorter) is equal to 720 pixels.
Thumbnail: A thumbnail is an image obtained after the original image is proportionally compressed. After the compression, the height or width (whichever is shorter) is equal to 198 pixels.
Sample code:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Image)
      {
         string path = elem.image_elem_orig_path; // Image upload path. This field
         switch(elem.image_elem_level)
         {
          case TIMImageLevel.kTIMImageLevel_Orig: // Send the original image
          {
            string id = elem.image_elem_orig_id;
            int h = elem.image_elem_orig_pic_height;
            int w = elem.image_elem_orig_pic_width;
            int size = elem.image_elem_orig_pic_size;
            string url = elem.image_elem_orig_url;
            break;
          }
          case TIMImageLevel.kTIMImageLevel_HD: // Send the large image (large in s
          {
            string id = elem.image_elem_large_id;
            int h = elem.image_elem_large_pic_height;
            int w = elem.image_elem_large_pic_width;
            int size = elem.image_elem_large_pic_size;
            string url = elem.image_elem_large_url;
            break;
          }
          case TIMImageLevel.kTIMImageLevel_Compression: // Send the thumbnail (sma
          {
            string id = elem.image_elem_thumb_id;
            int h = elem.image_elem_orig_thumb_height;
            int w = elem.image_elem_orig_thumb_width;
            int size = elem.image_elem_orig_thumb_size;
            string url = elem.image_elem_thumb_url;
            break;
```

```
            }
          }
        }
      }
    }
})
```

Video message

After a video message is received, a video preview is displayed on the chat page, and the video is played back after the user clicks the message.

Two steps are required:

The ChatSDK will download video messages when free, which can be directly used by you and will be cleared when the application is uninstalled.

The following sample code demonstrates how to parse the video content from `Message` :

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Video)
      {
       // Parse the video message attributes such as thumbnail, playback address, w
       string type = elem.video_elem_video_type;
       int size = elem.video_elem_video_size;
       int duration = elem.video_elem_video_duration;
       string path = elem.video_elem_video_path;
       string url = elem.video_elem_video_url;
       int imageType = elem.video_elem_image_type;
       int imageSize = elem.video_elem_image_size;
       int w = elem.video_elem_image_width;
       int h = elem.video_elem_image_height;
       string imagePath = elem.video_elem_image_path;
       string url = elem.video_elem_image_url;
      }
    }
  }
})
```

Audio message

The Chat SDK will download audio messages when free, which can be directly used by you and will be cleared when the application is uninstalled.

©2013-2025 Tencent Cloud International Pte. Ltd.

The following sample code demonstrates how to parse the audio content from `Message`:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Sound)
      {
       // Parse the audio playback address, local address, size, and duration
       int size = elem.sound_elem_file_size;
       int duration = elem.sound_elem_file_time;
       string path = elem.sound_elem_file_path;
       string url = elem.sound_elem_url;
      }
    }
  }
})
```

File message

The Chat SDK will download file messages when free, which can be directly used by you and will be cleared when the application is uninstalled.

The following sample code demonstrates how to parse the file content from `Message`:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_File)
      {
       // Parse the filename, size, URL, etc. of the file message
       int size = elem.file_elem_file_size;
       string path = elem.file_elem_file_path;
       string name = elem.file_elem_file_name;
       string url = elem.file_elem_url;
      }
    }
  }
})
```

Geographical location message

The following sample code demonstrates how to parse the geographical location content from `Message`:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Location)
      {
       // Parse the geographical location information such as latitude, longitude,
       double longitude = elem.location_elem_longitude;
       double latitude = elem.location_elem_latitude;
       string desc = elem.location_elem_desc;
      }
    }
  }
})
```

Emoji message

The SDK provides a passthrough channel only for emoji messages. Here, `index` and `data` can be customized. For example, the sender can set `index` to `1` and `data` to `x12345` to indicate the smile emoji.
The receiver parses the received emoji message as `1` and `x12345` and displays the message as the smile emoji according to the preset rules.
The following sample code demonstrates how to parse the emoji content from `Message`:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
  foreach(Message message in messages)
  {
    foreach (Elem elem in message.message_elem_array)
    {
      // There is a next element
      if (elem.elem_type == TIMElemType.kTIMElem_Face)
      {
       string data = elem.face_elem_buf;
       int index = elem.face_elem_index;
      }
    }
  }
})
```

Group tip message

Group tip messages are tips received by users in addition to ordinary messages in a group chat, for example, "The admin removed alice from the group chat" and "bob renamed the group "xxxx"".

Note:

Group tip messages will be received by only group members but not one-to-one chat parties.

There are many types of group tip messages. For more information, see the definition of `GroupTipsElem` (Details).

After receiving a group tip message, the receiver generally needs to:

1. Parse each field in `GroupTipsElem` .
2. Identify the message type according to `group_tips_elem_tip_type` .
3. Merge other fields into the content for display according to the type.

For example:

The receiver parses `type` as `kTIMGroupTip_GroupInfoChange` , indicating that this is a notification of group profile change.

The receiver can get the operator information from `group_tips_elem_op_user_info` and the modified group name from `group_tips_elem_group_change_info_array` .

At this point, the receiver can merge the "operator" and "modified group name" to create a group tip, such as "Alice renamed the group "group123"".

The following sample code demonstrates how to parse the tip content from `V2TIMMessage` :

```
if(tip.group_tips_elem_tip_type == TIMGroupTipType.kTIMGroupTip_GroupInfoChange){
        tip.group_tips_elem_group_id; // Group
        message.group_tips_elem_tip_type; // Group tip type
        message.group_tips_elem_op_user_info; // Operator profile
        message.group_tips_elem_changed_user_info_array; // User profile operated o
        message.group_tips_elem_group_change_info_array; // Group information chang
        message.group_tips_elem_member_change_info_array; // Group member change in
}
```

Getting the Message Sending Progress

```
TencentIMSDK.SetMsgElemUploadProgressCallback((Message message, int index, int cur_
    // `message` indicates the message instance being sent.
  // `index` indicates the number of the file being uploaded.
  // `cur_size` indicates the current size (in MB) of the file being uploaded.
  // `total_size` indicates the total size of the file being uploaded.
})
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Receiving a Message Containing Multiple Element Objects

1. Use the `Message` object to parse `message_elem_array` .
2. Traverse `message_elem_array` to get all the element objects.

Sample code:

```
foreach (Elem elem in message.message_elem_array) {
   // There is a next element
}
```

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

Last updated：2025-01-07 15:20:40

Feature Overview

To receive messages, you need to listen for the `MESSAGE_RECEIVED` event.

Listening for an Event

Note:

Call this API to listen for events before calling the `login` API in order to avoid missing the events delivered by the SDK.

API

```
chat.on(eventName, handler, context);
```

Parameters

Name	Type	Description
eventName	String	Event name. All event names are stored in the `TencentCloudChat.EVENT` variable. To view all the events, use `console.log(TencentCloudChat.EVENT)` .
handler	Function	Event handling method. When an event is triggered, this handler will be called to handle it.
context	* | undefined	The context expected for handler execution

Examples

```
let onMessageReceived = function(event) {
  // event.data - An array that stores `Message` objects - [Message]
  // For details of Message, please refer to https://www.tencentcloud.com/document/
  const messageList = event.data;
  messageList.forEach((message) => {
    if (message.type === TencentCloudChat.TYPES.MSG_TEXT) {
      // Text message
    } else if (message.type === TencentCloudChat.TYPES.MSG_IMAGE) {
      // Image message
    } else if (message.type === TencentCloudChat.TYPES.MSG_AUDIO) {
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
      // Audio message
    } else if (message.type === TencentCloudChat.TYPES.MSG_VIDEO) {
      // Video message
    } else if (message.type === TencentCloudChat.TYPES.MSG_FILE) {
      // File message
    } else if (message.type === TencentCloudChat.TYPES.MSG_CUSTOM) {
      // Custom message
    } else if (message.type === TencentCloudChat.TYPES.MSG_MERGER) {
      // Merged message
    } else if (message.type === TencentCloudChat.TYPES.MSG_LOCATION) {
      // Geographical location message
    } else if (message.type === TencentCloudChat.TYPES.MSG_GRP_TIP) {
      // Group tip message
    } else if (message.type === TencentCloudChat.TYPES.MSG_GRP_SYS_NOTICE) {
      // Group system notification
    }
  });
};
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

Canceling Event Listening

API

```
chat.off(eventName, handler, context);
```

Parameters

Name	Type	Description
eventName	String	Event name. All event names are stored in the `TencentCloudChat.EVENT` variable. To view all the events, use `console.log(TencentCloudChat.EVENT)` .
handler	Function	Event handling method. When an event is triggered, this handler will be called to handle it.
context	* | undefined	The context expected for handler execution

Examples

```
let onMessageReceived = function(event) {
  // event.name - TencentCloudChat.EVENT.MESSAGE_RECEIVED
  // event.data - An array that stores `Message` objects - [Message]
```

```
};
chat.off(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

Historical Message Android&iOS&Windows&Mac

Last updated：2025-01-13 17:27:05

Feature Description

The API for pulling historical messages is in the `V2TIMManager` and `V2TIMManager+Message (for Objective-C)` and `V2TIMMessageManager (Java and C++)` classes.
In addition to the support for pulling historical one-to-one and group messages, an advanced API is provided to pull messages by sequence, start point, or time range.
Both local and cloud historical messages can be pulled.
Note:
When historical messages are pulled from the cloud and a network exception is noticed, the SDK will return the locally stored historical messages.
Locally stored historical messages are not subject to time limits, but those stored in the cloud are subject to the following time limits:
Free Trail: The free storage period is 7 days and cannot be extended.
Standard edition: The free storage period is 7 days and can be extended.
Pro edition: The free storage period is 30 days and can be extended.
Pro Plus edition: The free storage period is 90 days and can be extended.
Enterprise edition: The free storage period is 90 days and can be extended.
Note:
It is a value-added service to extend the storage period of historical messages. If you need this feature, please contact us in Telegram technical group.
Rich media messages (such as images, files, and audios) have the same storage periods as historical messages.

Pulling One-to-One Messages

Call the `getC2CHistoryMessageList` API (Java / Swift / Objective-C) to get historical one-to-one messages.
When the network is normal, the latest cloud data will be pulled; when it is abnormal, the SDK will return the locally stored historical messages.
If you want to pull only local historical messages, see Advanced API.
This API supports pulling by page. For more information, see Pulling by page.
Sample code:

©2013-2025 Tencent Cloud International Pte. Ltd.

Java

Swift

Objective-C

C++

```
// Pull historical one-to-one messages
// Set `lastMsg` to `null` for the first pull
// `lastMsg` can be the last message in the returned message list for the second pu
V2TIMManager.getMessageManager().getC2CHistoryMessageList(#your user id#, 20, null,
    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "fail, " + code + ", " + desc);
    }

    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // Record the `lastMsg` for the next pull
        V2TIMMessage lastMsg = v2TIMMessages.get(v2TIMMessages.size() - 1);
        Log.i("imsdk", "success");
    }
});
```

```
// Pull historical one-to-one messages
// Set `lastMsg` to `null` for the first pull
// `lastMsg` can be the last message in the returned message list for the second pu
V2TIMManager.shared.getC2CHistoryMessageList(userID: "userID", count: 100, lastMsg:
    msgs.forEach { item in
        print("\\(item.description)")
    }
        // Record the `lastMsg` for the next pull
    lastMessage = msgs.last
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
// Pull historical one-to-one messages
// Set `lastMsg` to `nil` for the first pull
// `lastMsg` can be the last message in the returned message list for the second pu
[V2TIMManager.sharedInstance getC2CHistoryMessageList:#your user id# count:20 lastM
    // Record the `lastMsg` for the next pull
    V2TIMMessage *lastMsg = msgs.lastObject;
    NSLog(@"success, %@", msgs);
} fail:^(int code, NSString *desc) {
    NSLog(@"fail, %d, %@", code, desc);
}];
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Pulling Group Messages

Call the `getGroupHistoryMessageList` API (Java / Swift / Objective-C) to get historical group messages. When the network is normal, the latest cloud data will be pulled; when it is abnormal, the SDK will return the locally stored historical messages.

If you want to pull only local historical messages, see Advanced API.

This API supports pulling by page. For more information, see Pulling by page.

Note:

Only the historical messages of meeting groups (Meeting) can be pulled. For more information on group message limits, see Differences in message capabilities.

This API does not apply to audio-video groups (AVChatRoom), as their messages are not stored on the cloud roaming server or in the local database. If you use the Pro edition、Pro Plus edition 、Enterprise edition, you can configure **Message History for New Members** (Path：Applications > Your App > Chat > Configuration > Group Configuration > Group feature configuration > Message History for New Members > Previous Messages Viewable) in the Console to get historical messages in the `onRecvNewMessage` callback after successfully joining the audio-video group. New members of an audio-video group can view up to 20 messages in the past 24 hours before they join the group.

Sample code:

Java

Swift

Objective-C

C++

```
// Pull historical group messages
// Set `lastMsg` to `null` for the first pull
// `lastMsg` can be the last message in the returned message list for the second pu
V2TIMManager.getMessageManager().getGroupHistoryMessageList(#your group id#, 20, nu
    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "fail, " + code + ", " + desc);
    }

    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // Record the `lastMsg` for the next pull
        V2TIMMessage lastMsg = v2TIMMessages.get(v2TIMMessages.size() - 1);
        Log.i("imsdk", "success");
    }
```

```
});
```

```
// Pull historical group messages
// Set `lastMsg` to `null` for the first pull
// `lastMsg` can be the last message in the returned message list for the second pu
V2TIMManager.shared.getGroupHistoryMessageList(groupID: "groupID", count: 100, last
    msgs.forEach { item in
        print("\\(item.description)")
    }
     // Record the `lastMsg` for the next pull
    lastMessage = msgs.last
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
// Pull historical group messages
// Set `lastMsg` to `null` for the first pull
// `lastMsg` can be the last message in the returned message list for the second pu
[V2TIMManager.sharedInstance getGroupHistoryMessageList:#your group id# count:20 la
    // Record the `lastMsg` for the next pull
    V2TIMMessage *lastMsg = msgs.lastObject;
    NSLog(@"success, %@", msgs);
} fail:^(int code, NSString *desc) {
    NSLog(@"fail, %d, %@", code, desc);
}];
```

Advanced Features

Advanced API

If the ordinary API above cannot meet your needs to pull historical messages, you can use the advanced API `getHistoryMessageList` (Java / Swift / Objective-C / C++).

In addition to pulling historical one-to-one and group messages, this API supports the following advanced features:

Set the start point for message pull

Set the time range for message pull

Set the source for message pull: pull from the local database or the cloud

Specify the sequence for message pull: pull in reverse chronological order or in chronological order

Specify the message type for local pull: text, image, audio, video, file, emoji, group tip, merged, or custom message.

API prototype:

©2013-2025 Tencent Cloud International Pte. Ltd.

Java
Swift
Objective-C
C++

```
public abstract void getHistoryMessageList(V2TIMMessageListGetOption option, V2TIMV
```

```
public func getHistoryMessageList(option: V2TIMMessageListGetOption, succ: V2TIMMes
```

```
- (void)getHistoryMessageList:(V2TIMMessageListGetOption *)option
                         succ:(V2TIMMessageListSucc)succ
                         fail:(V2TIMFail)fail;
```

```
virtual void GetHistoryMessageList(const V2TIMMessageListGetOption& option,
                                   V2TIMValueCallback<V2TIMMessageVector>* callback
```

The parameters of the `V2TIMMessageListGetOption` class are as described below:

Parameter	Description	Valid for One-to-One Chat	Valid for Group Chat	Required	Remarks
getType	Source and sequence of the message pull, which can be set to **local/cloud** and **reverse chronological order/chronological order** respectively.	YES	YES	YES	When the pull source is set to the cloud, the local message list and cloud message list will be merge returned. If there is no network connection, the local message lis be returned.
userID	The specified user ID with which to pull historical one-to-one messages	YES	NO	NO	To pull one-to-one messages wit certain user, you need to specify the `userID` .
groupID	The specified group ID with which to pull historical group messages	NO	YES	NO	To pull group messages from a certain group, you need to specif only the `groupID` .
count	Number of messages	YES	YES	YES	We recommend you set it to `20`

©2013-2025 Tencent Cloud International Pte. Ltd.

	per pull				otherwise, the pull speed may be affected.
messageTypeList	Pulled message type set	YES	YES	NO	1. It is supported only for local pu that is, it is valid only when getTy `V2TIM_GET_LOCAL_OLDER_M` or `V2TIM_GET_LOCAL_NEWER_M` 2. If this field is left empty, messa of all types will be pulled. 3. For more information on the supported message types, see V2TIMElemType (Java / Swift / Objective-C / C++).
lastMsg	Last message	YES	YES	NO	This parameter applies to historic message pull. 1. It can be used for both one-to- and group chats. 2. If it is set as the start point for message pull, the message will **r be included** in the returned mes list. 3. If it is left empty, the latest mes in the conversation will be used a start point for pull.
lastMsgSeq	`seq` of the last message	NO	YES	NO	This parameter applies to historic message pull or locating. 1. It can be used only for group c 2. If it is set as the start point for message pull, the message will b **included** in the returned messag list. 3. If both `lastMsg` and `lastMsgSeq` are specified, th SDK will use `lastMsg` . 4. If neither `lastMsg` nor `lastMsgSeq` is specified, the point for pull will be determined b on whether the `getTimeBegi` set. If yes, the set range will be u as the start point; if no, the latest message will be used as the star point.

©2013-2025 Tencent Cloud International Pte. Ltd.

getTimeBegin	Start time for the message pull. It is a UTC timestamp in seconds.	YES	YES	NO	It defaults to `0`, indicating to p messages from the current time.
getTimePeriod	Time range for the message pull in seconds.	YES	YES	NO	1. It defaults to `0`, indicating th there is no time range limit. 2. It is a closed range including tl start and end time: If `getType` specifies the reve chronological order, the time ran [`getTimeBegin` - `getTimePeriod`, `getTimeBegin`]. If `getType` specifies the chronological order, the time ran [`getTimeBegin`, `getTimeBegin` + `getTimePeriod`].

Pulling by page

Historical one-to-one message pull, historical group message pull, and pull via the advanced API can all be paged by using `lastMsg` and `count`:

1. For the first pull, leave `lastMsg` empty. In this case, the SDK will pull the latest message.

2. If it is not the first pull, use the last message in the message list from the last pull as the `lastMsg`. In this case, the `lastMsg` will not be included in the returned message list.

3. If the message list returned in the successful callback is empty, it means that pulling completed and there is no more data.

Note:

1. In the pulled message list, newer messages are more forward.

2. We recommend you set `count` to `20` to enhance the loading efficiency and save network traffic.

If you use `lastMsgSeq` to pull historical messages, the message corresponding to `lastMsgSeq` will be **included** in the returned message list.

Therefore, we recommend you not use `lastMsgSeq` if you are not pulling historical group messages for the first time (subsequent pull); otherwise, the same message may be pulled repeatedly.

For example, there are eight historical messages: `msg1`, `msg2`, `msg3`, `msg4`, `msg5`, `msg6`, `msg7`, and `msg8`.

If four messages are pulled each time, you will get `msg1`, `msg2`, `msg3`, and `msg4` for the first pull. If you use the `lastMsgSeq` of `msg4` to start another pull, `msg4`, `msg5`, `msg6`, and `msg7` will be pulled. In

©2013-2025 Tencent Cloud International Pte. Ltd.

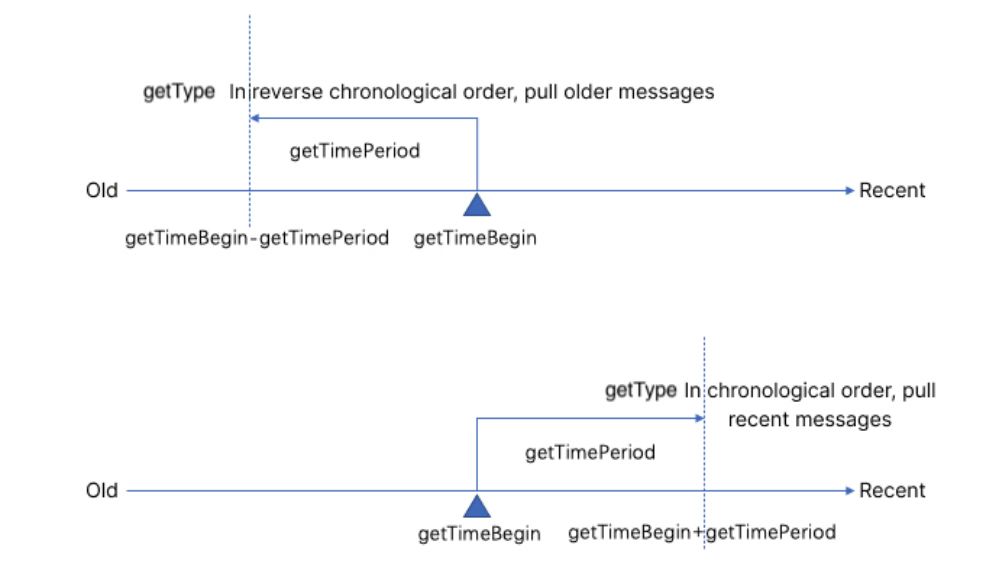
this case, `msg4` is pulled twice.

If you need to use `lastMsgSeq` for subsequent pulls, we recommend you set the logic of message deduplication.

Pulling by time range

You can set `getTimeBegin` and `getTimePeriod` to specify the time range for the message pull.

The start and end timestamps of the time range are related to `getType` , as shown below:



The following sample code demonstrates the process of pulling cloud group messages of the whole day from 2022-01-01 00:00:00 (the timestamp is `1640966400`) in reverse chronological order.

Java

Swift

Objective-C

C++

```
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // Pull old
option.setGetTimeBegin(1640966400);          // Start from 2022-01-01 00:00:00
option.setGetTimePeriod(1 * 24 * 60 * 60);   // Pull the messages of the whole day
option.setCount(Integer.MAX_VALUE);          // Return all the messages within the t
option.setGroupID(#you group id#);           // Pull group messages
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
    }

    @Override
```

```
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
let option = V2TIMMessageListGetOption()
option.getType = .V2TIM_GET_CLOUD_OLDER_MSG// Pull older cloud messages
option.getTimeBegin = 1640966400 // Start from 2022-01-01 00:00:00
option.getTimePeriod = 1 * 24 * 60 * 60  // Pull the messages of the whole day
option.groupID = "#your group id#"
option.count = UInt(INT_MAX)   // Return all the messages within the time range
V2TIMManager.shared.getHistoryMessageList(option: option) { msgs in
    msgs.forEach { item in

    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
V2TIMMessageListGetOption *option = [[V2TIMMessageListGetOption alloc] init];
option.getType = V2TIM_GET_CLOUD_OLDER_MSG; // Pull cloud messages in reverse chron
option.getTimeBegin = 1640966400;        // Start from 2022-01-01 00:00:00
option.getTimePeriod = 1 * 24 * 60 * 60; // Pull the messages of the whole day
option.count = INT_MAX;                  // Return all the messages within the time
option.groupID = #your group id#;        // Pull group messages
[V2TIMManager.sharedInstance getHistoryMessageList:option succ:^(NSArray<V2TIMMessa
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }
```

```
    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageListGetOption option;
option.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_OLDER_MSG;  // Pull older clo
option.getTimeBegin = 1640966400;        // Start from 2022-01-01 00:00:00
option.getTimePeriod = 1 * 24 * 60 * 60;                          // Pull the messa
option.count = std::numeric_limits<uint64_t>::max();              // Return all the
option.groupID = "your group id";                                 // Pull group mes

auto callback = new ValueCallback<V2TIMMessageVector>{};
callback->SetCallback(
    [=](const V2TIMMessageVector& messageList) {
        std::cout << "success" << std::endl;
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(option, cal
```

Pulling by page within the time range

You can specify both the time range and `lastMsg` / `lastMsgSeq` for the message pull at the same time. And the SDK will respond as follows:

1. If both `getTimeBegin` / `getTimePeriod` and `lastMsg` / `lastMsgSeq` are set, the overlapping part of the messages pulled based on the start message and the messages pulled based on the time range will be returned.
2. If neither `getTimeBegin` / `getTimePeriod` nor `lastMsg` / `lastMsgSeq` is set, messages will be pulled from the latest message based on the sequence and method specified by `getType` .

©2013-2025 Tencent Cloud International Pte. Ltd.

The following sample code demonstrates the process of pulling cloud group messages by page (20 messages per page) from 2022-01-01 00:00:00 (the timestamp is `1640966400`) in reverse chronological order.

Java

Swift

Objective-C

C++

```
// Define a variable to record the cursor of each pull
private V2TIMMessage m_lastMsg = null;  // It is `null` for the first pull.

// Logic of pulling by page
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // Pull old
option.setGetTimeBegin(1640966400);          // Start from 2022-01-01 00:00:00
option.setGetTimePeriod(1 * 24 * 60 * 60);  // Pull the messages of the whole day
option.setCount(20);                         // 20 messages per page
option.setLastMsg(m_lastMsg);                // Position of the last pull (the last
option.setGroupID(#you group id#);           // Pull group messages
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
        // Record the `lastMsg` for the next pull
        m_lastMsg = v2TIMMessages.get(v2TIMMessages.size() - 1);
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
// Define a variable to record the cursor of each pull
var lastMsg: V2TIMMessage?

// Logic of pulling by page
let option = V2TIMMessageListGetOption()
option.getType = .V2TIM_GET_CLOUD_OLDER_MSG // Pull cloud messages in reverse chron
option.getTimeBegin = 1640966400 // Start from 2022-01-01 00:00:00
option.getTimePeriod = 1 * 24 * 60 * 60  // Pull the messages of the whole day
option.count = 20    // 20 messages per page
option.lastMsg = self.lastMsg;    // Position of the last pull (the last message in
option.groupID = "#your group id#"   // Pull group messages
V2TIMManager.shared.getHistoryMessageList(option: option) { msgs in
    msgs.forEach { item in
```

```
    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
// Define a variable to record the cursor of each pull
@property (nonatomic, copy) V2TIMMessage *lastMsg;

// Logic of pulling by page
V2TIMMessageListGetOption *option = [[V2TIMMessageListGetOption alloc] init];
option.getType = V2TIM_GET_CLOUD_OLDER_MSG; // Pull cloud messages in reverse chron
option.getTimeBegin = 1640966400;        // Start from 2022-01-01 00:00:00
option.getTimePeriod = 1 * 24 * 60 * 60; // Pull the messages of the whole day
option.count = 20;                       // 20 messages per page
option.lastMsg = self.lastMsg;           // Position of the last pull (the last mes
option.groupID = #your group id#;        // Pull group messages
__weak typeof(self) weakSelf = self;
[V2TIMManager.sharedInstance getHistoryMessageList:option succ:^(NSArray<V2TIMMessa
    NSLog(@"success");
    // Record the `lastMsg` for the next pull
    weakSelf.lastMsg = msgs.lastObject;
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
```

```
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

// Define a variable to record the cursor of each pull
V2TIMMessage lastMsg;

// Logic of pulling by page
V2TIMMessageListGetOption option;
option.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_OLDER_MSG;  // Pull older clo
option.getTimeBegin = 1640966400;        // Start from 2022-01-01 00:00:00
option.getTimePeriod = 1 * 24 * 60 * 60;                          // Pull the messa
option.count = 20;                                                // 20 messages pe
option.lastMsg = First pull? nullptr : &lastMsg;  // Position of the last pull (the
option.groupID = "your group id";  // Pull group messages

auto callback = new ValueCallback<V2TIMMessageVector>{};
callback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        std::cout << "success" << std::endl;
        // Record the `lastMsg` for the next pull
        lastMsg = messageList[messageList.Size() - 1];
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(option, cal
```

Pulling only local messages

Set `getType` to pull only local messages:

When `getType` is set to `V2TIM_GET_LOCAL_OLDER_MSG`, locally stored messages will be pulled in reverse chronological order.

When `getType` is set to `V2TIM_GET_LOCAL_NEWER_MSG`, locally stored messages will be pulled in chronological order.

©2013-2025 Tencent Cloud International Pte. Ltd.

The following sample code demonstrates the process of pulling 20 one-to-one messages from the local database in reverse chronological order, starting from the latest message:

Java

Swift

Objective-C

C++

```
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_LOCAL_OLDER_MSG); // Pull loc
option.setLastMsg(null);            // Set to pull from the latest message
option.setCount(20);                // Pull 20 messages
option.setUserID(#you user id#);    // Pull one-to-one messages
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
let option = V2TIMMessageListGetOption()
option.getType = .V2TIM_GET_LOCAL_OLDER_MSG // Pull local messages in reverse chron
option.lastMsg = nil // Set to pull from the latest message
option.count = 20    // Pull 20 messages
option.userID = "userID" // Pull one-to-one messages
V2TIMManager.shared.getHistoryMessageList(option: option) { msgs in
    msgs.forEach { item in

    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
V2TIMMessageListGetOption *option = [[V2TIMMessageListGetOption alloc] init];
option.getType = V2TIM_GET_LOCAL_OLDER_MSG; // Pull local messages in reverse chron
option.lastMsg = nil;            // Set to pull from the latest message
option.count = 20;               // Pull 20 messages
option.userID = #your user id#;  // Pull one-to-one messages
[V2TIMManager.sharedInstance getHistoryMessageList:option succ:^(NSArray<V2TIMMessa
    NSLog(@"success");
```

```
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageListGetOption option;
option.getType = V2TIMMessageGetType::V2TIM_GET_LOCAL_OLDER_MSG;  // Pull older loc
option.count = 20;                                                // Pull 20 messag
option.userID = "you user id";                                    // Pull one-to-on

auto callback = new ValueCallback<V2TIMMessageVector>{};
callback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        std::cout << "success" << std::endl;
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
```

```
        std::cout << "error" << std::endl;
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(option, cal
```

Pulling messages of a specified type

The SDK defines common message types, such as text, image, and video messages. For more information, see `V2TIMElemType` (Java / Swift / Objective-C / C++). The advanced API `getHistoryMessageList` allows setting `messageTypeList` to specify the message type for pull.

Note:

1. When `messageTypeList` is left empty, messages of all types will be pulled.
2. The message type can be specified only for the local pull but not for the cloud pull.

The following sample code demonstrates the process of pulling 20 text and image messages in reverse chronological order, starting from the current time:

Java

Swift

Objective-C

C++

```
// Pull image and text messages
ArrayList<Integer> messageTypeList = new ArrayList<Integer>();
messageTypeList.add(V2TIM_ELEM_TYPE_IMAGE);
messageTypeList.add(V2TIM_ELEM_TYPE_TEXT);

V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // Pull old
option.setCount(20);
option.setMessageTypeList(messageTypeList);
option.setGroupID("you group id");
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
let option = V2TIMMessageListGetOption()
option.getType = .V2TIM_GET_LOCAL_OLDER_MSG
option.messageTypeList = [.V2TIM_ELEM_TYPE_IMAGE, .V2TIM_ELEM_TYPE_TEXT]
option.count = 20
option.groupID = "your group id";
V2TIMManager.shared.getHistoryMessageList(option: option) { msgs in
    msgs.forEach { item in

    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
V2TIMMessageListGetOption *option = [[V2TIMMessageListGetOption alloc] init];
option.getType = V2TIM_GET_CLOUD_OLDER_MSG;
// Pull image and text messages
option.messageTypeList = @[@(V2TIM_ELEM_TYPE_IMAGE), @(V2TIM_ELEM_TYPE_TEXT)];
option.count = 20;
option.groupID = @"your group id";
[V2TIMManager.sharedInstance getHistoryMessageList:option succ:^(NSArray<V2TIMMessa
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
```

```
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageListGetOption option;
option.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_OLDER_MSG;  // Pull older clo
option.count = 20;                                                // Pull 20 messag
option.messageTypeList.PushBack(V2TIMElemType::V2TIM_ELEM_TYPE_IMAGE);  // Pull ima
option.messageTypeList.PushBack(V2TIMElemType::V2TIM_ELEM_TYPE_TEXT);   // Pull tex
option.userID = "you group id";                                   // Pull gro

auto callback = new ValueCallback<V2TIMMessageVector>{};
callback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        std::cout << "success" << std::endl;
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(option, cal
```

Pulling messages of a specified sequence

When polling historical messages in group chats, you can poll according to the specified message sequence.

Note:

It is only valid when pulling group history messages;

The message sequence can be obtained through the `seq` field of the V2TIMMessage object;

When `getType` is set to poll from the cloud, the local stored message list and the cloud stored message list will be merged and returned; if there is no network, the local message list will be returned directly;

When `getType` is set to poll from local, the local message list will be returned directly;

When `getType` is set to poll older messages, the message list is in reverse chronological order, i.e., messages are sorted from large to small by timestamp;

When `getType` is set to poll updated messages, the message list is in chronological order, i.e., messages are sorted from small to large by timestamp.

©2013-2025 Tencent Cloud International Pte. Ltd.

The sample code will demonstrate: In the group, poll messages with sequence 1, 3, 5, and 9 from the cloud, and return the messages in reverse chronological order.

Java

Swift

Objective-C

C++

```
ArrayList<Long> sequenceList = new ArrayList<>();
sequenceList.add(1L);
sequenceList.add(3L);
sequenceList.add(5L);
sequenceList.add(9L);
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGroupID("your groupID");
option.setMessageSeqList(sequenceList);
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG;);
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {

    }

    @Override
    public void onError(int code, String desc) {

    }
});
```

```
option.getType = .V2TIM_GET_LOCAL_OLDER_MSG
option.groupID = "your group id"
option.messageSeqList = [1,3,5,9]
V2TIMManager.shared.getHistoryMessageList(option: option) { msgs in
    msgs.forEach { item in

    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
V2TIMMessageListGetOption *option = [[V2TIMMessageListGetOption alloc] init];
option.getType = V2TIM_GET_CLOUD_OLDER_MSG;
option.groupID = @"your groupID";
option.messageSeqList = @[@(1), @(3), @(5), @(9)];
[V2TIMManager.sharedInstance getHistoryMessageList:option succ:^(NSArray<V2TIMMessa
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
} fail:^(int code, NSString *desc) {

}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMUInt64Vector sequence_list;
sequence_list.PushBack(1);
sequence_list.PushBack(3);
sequence_list.PushBack(5);
sequence_list.PushBack(9);
V2TIMMessageListGetOption option;
option.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_OLDER_MSG;
option.messageSeqList = sequence_list;
option.groupID = "your group id";

auto callback = new ValueCallback<V2TIMMessageVector>{};
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
callback->SetCallback(
    [=](const V2TIMMessageVector& messageList) {
        std::cout << "success" << std::endl;
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(option, cal
```

Quick redirection to a group @ message

After a user receives a group @ message in a group conversation, the user generally needs to click the @ bar to go to the message and pull the neighboring messages for display.

As the group @ message also needs to be displayed, you can set its `sequence` as the `lastMsgSeq` and use the advanced API `getHistoryMessageList` to pull messages.

The following sample code demonstrates the process of clicking the @ bar, redirecting to the group @ message, and pulling the first 20 messages earlier and later than that message for display:

Java

Swift

Objective-C

C++

```
// Get the `sequence` of the group @ message
long atSequence = 1081;

// Pull the group @ message and earlier messages
V2TIMMessageListGetOption beforeOption = new V2TIMMessageListGetOption();
beforeOption.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // Pu
beforeOption.setCount(20);                          // Pull 20 messages
beforeOption.setLastMsgSeq(atSequence);             // Start the pull from the group
beforeOption.setGroupID(#you group id#);            // Pull group messages
V2TIMManager.getMessageManager().getHistoryMessageList(beforeOption, new V2TIMValue
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // The group @ message is included in the returned message list.
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
```

```
    }
});

// Pull messages later than the group @ message
V2TIMMessageListGetOption afterOption = new V2TIMMessageListGetOption();
afterOption.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_NEWER_MSG); // Pul
afterOption.setCount(20);                          // Pull 20 messages
afterOption.setLastMsgSeq(atSequence + 1);         // Start the pull from the first m
afterOption.setGroupID(#you group id#);            // Pull group messages
V2TIMManager.getMessageManager().getHistoryMessageList(afterOption, new V2TIMValueC
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // The group @ message is not included in the returned message list.
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
// Get the `sequence` of the group @ message
let  atSequence = 1081

// Pull the group @ message and earlier messages
let beforeOption = V2TIMMessageListGetOption()
beforeOption.getType = .V2TIM_GET_CLOUD_OLDER_MSG // Pull messages earlier than the
beforeOption.count = 20  // Pull 20 messages
beforeOption.lastMsgSeq = atSequence // Start the pull from the group @ message, wh
beforeOption.groupID = "your group id"; // Pull group messages
V2TIMManager.shared.getHistoryMessageList(option: beforeOption) { msgs in
    msgs.forEach { item in
        // The group @ message is included in the returned message list.
    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}

// Pull messages later than the group @ message
let afterOption = V2TIMMessageListGetOption()
afterOption.getType = .V2TIM_GET_CLOUD_NEWER_MSG // Pull messages later than the gr
afterOption.count = 20 // Pull 20 messages
afterOption.lastMsgSeq = atSequence + 1; // Start the pull from the first message l
afterOption.groupID = "your group id"; // Pull group messages
```

```
V2TIMManager.shared.getHistoryMessageList(option: afterOption) { msgs in
    msgs.forEach { item in
    // The group @ message is not included in the returned message list.
    }
    print("success")
} fail: { code, desc in
    print("fail \\(code), \\(desc)")
}
```

```
// Get the `sequence` of the group @ message
NSInteger atSequence = 1081;

// Pull the group @ message and earlier messages
V2TIMMessageListGetOption *beforeOption = [[V2TIMMessageListGetOption alloc] init];
beforeOption.getType = V2TIM_GET_CLOUD_OLDER_MSG; // Pull messages earlier than the
beforeOption.count = 20;                          // Pull 20 messages
beforeOption.lastMsgSeq = atSequence;             // Start the pull from the group
beforeOption.groupID = #your group id#;           // Pull group messages
[V2TIMManager.sharedInstance getHistoryMessageList:beforeOption succ:^(NSArray<V2TI
    // The group @ message is included in the returned message list.
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];

// Pull messages later than the group @ message
V2TIMMessageListGetOption *afterOption = [[V2TIMMessageListGetOption alloc] init];
afterOption.getType = V2TIM_GET_CLOUD_NEWER_MSG; // Pull messages later than the gr
afterOption.count = 20;                          // Pull 20 messages
afterOption.lastMsgSeq = atSequence + 1;         // Start the pull from the first m
afterOption.groupID = #your group id#;           // Pull group messages
[V2TIMManager.sharedInstance getHistoryMessageList:afterOption succ:^(NSArray<V2TIM
    // The group @ message is not included in the returned message list.
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;
```

```
    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

// Get the `sequence` of the group @ message
uint64_t atSequence = 1081;

// Pull the group @ message and earlier messages
V2TIMMessageListGetOption beforeOption;
beforeOption.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_OLDER_MSG;  // Pull mes
beforeOption.count = 20;                                                // Pull 20
beforeOption.lastMsgSeq = atSequence;             // Start the pull from the group
beforeOption.userID = "you group id";  // Pull group messages

auto beforeCallback = new ValueCallback<V2TIMMessageVector>{};
beforeCallback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        // The group @ message is included in the returned message list.
        std::cout << "success" << std::endl;
        delete beforeCallback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete beforeCallback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(beforeOptio

// Pull the group @ message and earlier messages
```

```
V2TIMMessageListGetOption afterOption;
afterOption.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_NEWER_MSG;  // Pull mess
afterOption.count = 20;                                                 // Pull 20
afterOption.lastMsgSeq = atSequence + 1;         // Start the pull from the first m
afterOption.userID = "you group id";       // Pull group messages

auto afterCallback = new ValueCallback<V2TIMMessageVector>{};
afterCallback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        // The group @ message is not included in the returned message list.
        std::cout << "success" << std::endl;
        delete afterCallback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete afterCallback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(afterOption
```

Message locating for searches

When a user gets the `V2TIMMessage` object after the local search and clicks the message, you need to redirect the user to the position of the message and display neighboring messages.

At this point, you can set `lastMsg` to the message object and use the advanced API `getHistoryMessageList` to pull neighboring messages.

The following sample code demonstrates the process of pulling the first 20 messages earlier and later than the located message for display.

Java

Swift

Objective-C

C++

```
// Get the current message object
V2TIMMessage lastMsg = #The located message#;

// Pull messages earlier than the specified message
V2TIMMessageListGetOption beforeOption = new V2TIMMessageListGetOption();
beforeOption.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // Pu
beforeOption.setCount(20);                          // Pull 20 messages
beforeOption.setLastMsg(lastMsg);                   // Start the pull from the specif
beforeOption.setGroupID(#you group id#);            // Pull group messages
V2TIMManager.getMessageManager().getHistoryMessageList(beforeOption, new V2TIMValue
    @Override
```

```
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // `lastMsg` is not included in the returned message list.
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});

// Pull messages later than the specified message
V2TIMMessageListGetOption afterOption = new V2TIMMessageListGetOption();
afterOption.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_NEWER_MSG); // Pul
afterOption.setCount(20);                          // Pull 20 messages
afterOption.setLastMsg(lastMsg);                   // Start the pull from the specifi
afterOption.setGroupID(#you group id#);            // Pull group messages
V2TIMManager.getMessageManager().getHistoryMessageList(afterOption, new V2TIMValueC
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // `lastMsg` is not included in the returned message list.
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

```
// Get the current message object
let lastMsg: V2TIMMessage = "The located message"

// Pull messages earlier than the specified message
let beforeOption = V2TIMMessageListGetOption()
beforeOption.getType = .V2TIM_GET_CLOUD_OLDER_MSG // Pull messages earlier than the
beforeOption.count = 20                            // Pull 20 messages
beforeOption.lastMsg = lastMsg                     // Start the pull from the specif
beforeOption.groupID = "your_group_id"           // Pull group messages

V2TIMManager.sharedInstance().getHistoryMessageList(beforeOption, succ: { msgs in
    // The specified message is not included in the returned message list.
    print("success")
}) { code, desc in
    print("failure, code: \\(code), desc: \\(desc)")
}
```

```
// Pull messages later than the specified message
let afterOption = V2TIMMessageListGetOption()
afterOption.getType = .V2TIM_GET_CLOUD_NEWER_MSG // Pull messages later than the sp
afterOption.count = 20                           // Pull 20 messages
afterOption.lastMsg = lastMsg                    // Start the pull from the specifi
afterOption.groupID = "your_group_id"           // Pull group messages

V2TIMManager.sharedInstance().getHistoryMessageList(afterOption, succ: { msgs in
    // The specified message is not included in the returned message list.
    print("success")
}) { code, desc in
    print("failure, code: \\(code), desc: \\(desc)")
}
```

```
// Get the current message object
V2TIMMessage *lastMsg = #The located message#;

// Pull messages earlier than the specified message
V2TIMMessageListGetOption *beforeOption = [[V2TIMMessageListGetOption alloc] init];
beforeOption.getType = V2TIM_GET_CLOUD_OLDER_MSG; // Pull messages earlier than the
beforeOption.count = 20;                          // Pull 20 messages
beforeOption.lastMsg = lastMsg;                   // Start the pull from the specif
beforeOption.groupID = #your group id#;           // Pull group messages
[V2TIMManager.sharedInstance getHistoryMessageList:beforeOption succ:^(NSArray<V2TI
    // The specified message is not included in the returned message list.
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];

// Pull messages later than the specified message
V2TIMMessageListGetOption *afterOption = [[V2TIMMessageListGetOption alloc] init];
afterOption.getType = V2TIM_GET_CLOUD_NEWER_MSG; // Pull messages later than the sp
afterOption.count = 20;                          // Pull 20 messages
afterOption.lastMsg = lastMsg;                   // Start the pull from the specifi
afterOption.groupID = #your group id#;           // Pull group messages
[V2TIMManager.sharedInstance getHistoryMessageList:afterOption succ:^(NSArray<V2TIM
    // The specified message is not included in the returned message list.
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
```

```
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

// Get the current message object
V2TIMMessage lastMsg = The located message;

// Pull messages earlier than the specified message
V2TIMMessageListGetOption beforeOption;
beforeOption.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_OLDER_MSG;  // Pull mes
beforeOption.count = 20;                                                // Pull 20
beforeOption.lastMsg = lastMsg;                    // Start the pull from the specif
beforeOption.userID = "you group id";  // Pull group messages

auto beforeCallback = new ValueCallback<V2TIMMessageVector>{};
beforeCallback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        // `lastMsg` is not included in the returned message list.
        std::cout << "success" << std::endl;
        delete beforeCallback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
```

```
        delete beforeCallback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(beforeOptio

// Pull messages later than the specified message
V2TIMMessageListGetOption afterOption;
afterOption.getType = V2TIMMessageGetType::V2TIM_GET_CLOUD_NEWER_MSG;  // Pull mess
afterOption.count = 20;                                                 // Pull 20
afterOption.lastMsg = lastMsg;                    // Start the pull from the specifi
afterOption.userID = "you group id";       // Pull group messages

auto afterCallback = new ValueCallback<V2TIMMessageVector>{};
afterCallback->SetCallback(
    [=](const V2TIMMessageVector& messageList) mutable {
        // `lastMsg` is not included in the returned message list.
        std::cout << "success" << std::endl;
        delete afterCallback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        std::cout << "error" << std::endl;
        delete afterCallback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->GetHistoryMessageList(afterOption
```

Parsing a signaling message

If you use the SDK's signaling API (Java / Swift /Objective-C), when you call `invite/inviteInGroup` and set `onlineUserOnly` to `false` (Java) / `NO` (iOS), a custom message will be generated for each signaling operation (including `invite`, `cancel`, `accept`, `reject`, and `timeout`). The message will be delivered to the user in the `onRecvNewMessage` callback of `V2TIMAdvancedMsgListener` and can also be pulled from historical messages.

If you want to customize the display text based on the signaling information, you can use the signaling parsing API `getSignalingInfo` (Java / Swift / Objective-C / C++) to get the signaling information.

API prototype

Java

Swift

Objective-C

C++

```
/**
 * Get the signaling information
 *
```

```
 * If `onlineUserOnly` is set to `false` in the `invite` operation, a custom messag
 The message will be delivered to users via `V2TIMAdvancedMsgListener` -> `onRecvNe
 *
 * @param msg // Message object
 * @return V2TIMSignalingInfo //Signaling information. If the value is null, `msg`
 */
public abstract V2TIMSignalingInfo getSignalingInfo(V2TIMMessage msg);
```

```
/**
 * Get the signaling information
 *
 * If `onlineUserOnly` is set to `NO` in the `invite` operation, a custom message w
 *
 *  @param msg // Message object
 *  @return V2TIMSignalingInfo //Signaling information. If the value is nil, `msg`
 */

public func getSignallingInfo(msg: V2TIMMessage) -> V2TIMSignalingInfo?
```

```
/**
 * Get the signaling information
 *
 * If `onlineUserOnly` is set to `NO` in the `invite` operation, a custom message w
 *
 *  @param msg // Message object
 *  @return V2TIMSignalingInfo //Signaling information. If the value is nil, `msg`
 */
- (V2TIMSignalingInfo *)getSignallingInfo:(V2TIMMessage *)msg;
```

```
/**
 * Get the signaling information
 *
 * If `onlineUserOnly` is set to `false` in the `invite` operation,
 * a custom message is generated for each signaling operation (including the `Invit
 * `V2TIMAdvancedMsgListener` -> `onRecvNewMessage`
 * and can be pulled from historical messages. To customize the display text based
 *
 * @param msg // Message object
 * @return V2TIMSignalingInfo //Signaling information. If `V2TIMSignalingInfo::invi
 */
virtual V2TIMSignalingInfo GetSignalingInfo(const V2TIMMessage& msg) = 0;
```

The following sample code demonstrates the process of pulling historical messages and parsing the display text of the signaling message.

Java

Swift

Objective-C

C++

```
/// Custom text of the signaling message
public String getCallSignalingContentWithMessage(V2TIMMessage message) {
    if (message == null) {
        Log.e(TAG, " invalid param ");
        return "";
    }

    V2TIMSignalingInfo signalingInfo = V2TIMManager.getSignalingManager().getSignal
    if (signalingInfo == null) {
        Log.e(TAG, " signalingInfo is null ");
        return "";
    }

    String content = "";
    Gson gson = new Gson();
    Object businessIdObj = null;
    String businessId = null;
    try {
        HashMap signalDataMap = gson.fromJson(signalingInfo.getData(), HashMap.clas
        if (signalDataMap != null) {
            businessIdObj = signalDataMap.get(TUIConstants.Message.CUSTOM_BUSINESS_
        } else {
            Log.e(TAG, " signalDataMap is null ");
            return "";
        }
    } catch (JsonSyntaxException e) {
        Log.e(TAG, " get signalingInfoCustomJsonMap error ");
    }

    if (businessIdObj instanceof String) {
        businessId = (String) businessIdObj;
    }

    if (TextUtils.equals(businessId, "av_call")) {
        if (signalingInfo.getActionType() == V2TIMSignalingInfo.SIGNALING_ACTION_TY
            // Make a call
            content = "@Make a call";
        } else if (signalingInfo.getActionType() == V2TIMSignalingInfo.SIGNALING_AC
            // Answer the call
            content = "@Answer the call";
        } else {
            // Other
```

```
            // ...
        }
    }

    return content;
}
```

```
/// Custom text of the signaling message
class func getCallSignalingContent(with message: V2TIMMessage) -> String? {
    guard let info = V2TIMManager.shared.getSignallingInfo(msg: message) else {
        return nil
    }

    // Parse the `data` field passed through
    var param: [String: Any]?
    if let data = info.data {
        if let jsonData = data.data(using: .utf8) {
            do {
                param = try JSONSerialization.jsonObject(with: jsonData, options: .
            } catch {
                print("JSON parsing error: \\(error)")
            }
        }
    }

    guard let parameters = param else {
        return nil
    }

    // Identify the business type
    guard let businessId = parameters["businessID"] as? String else {
        return nil
    }

    // Determine whether it is an audio/video call signaling
    var content: String?
    if businessId == "av_call" {
        switch info.actionType {
        case .V2TIMSignalingActionType_Invite:
            // Make a call
            content = "Make a call"
        case .V2TIMSignalingActionType_Accept_Invite:
            // Answer the call
            content = "Answer the call"
        default:
            // Other
```

```
            // ...
            break
        }
    }

    return content
}
```

```
/// Custom text of the signaling message
+ (NSString *)getCallSignalingContentWithMessage:(V2TIMMessage *)message
{
    V2TIMSignalingInfo *info = [[V2TIMManager sharedInstance] getSignallingInfo:mes
    if (!info) {
        return nil
    }
    // Parse the `data` field passed through
    NSError *err = nil;
    NSDictionary *param = nil;
    if (info.data != nil) {
        param = [NSJSONSerialization JSONObjectWithData:[info.data dataUsingEncodin
    }
    if (!param || ![param isKindOfClass:[NSDictionary class]]) {
        return nil
    }

    // Identify the business type
    NSArray *allKeys = param.allKeys;
    if (![allKeys containsObject:@"businessID"]) {
        return nil
    }
    NSString *businessId = [param objectForKey:@"businessID"];

    // Determine whether it is an audio/video call signaling
    NSString *content = nil;
    if ([businessId isEqualToString:@"av_call"]) {
        if (info.actionType == SignalingActionType_Invite) {
            // Make a call
            content = @"Make a call";
        } else if (info.actionType == SignalingActionType_Accept_Invite) {
            // Answer the call
            content = @"Answer the call";
        } else {
            // Other
            // ...
        }
    }
```

```
    return content;
}
```

```
Custom text of the signaling message
std::string GetCallSignalingContentWithMessage(const V2TIMMessage& message) {
    V2TIMSignalingInfo signalingInfo =
        V2TIMManager::GetInstance()->GetSignalingManager()->GetSignalingInfo(message
    if (signalingInfo.inviteID.Empty()) {
        // If `V2TIMSignalingInfo::inviteID` is empty, `message` is not a signaling
        return {};
    }

    // Parse the `data` field passed through, using the open source library RapidJSO
    rapidjson::Document document;
    document.Parse(signalingInfo.data.CString(), signalingInfo.data.Size());
    if (!document.HasMember("businessId") || !document["businessId"].IsString()) {
        return {};
    }
    const char* businessId = document["businessId"].GetString();

    // Determine whether it is an audio/video call signaling
    std::string content;
    if (businessId == "av_call") {
        if (signalingInfo.actionType == V2TIMSignalingActionType::SignalingActionTyp
            // Make a call
            content = u8"Make a call";
        } else if (signalingInfo.actionType == 2TIMSignalingActionType::SignalingAct
            // Answer the call
            content = u8"Answer the call";
        } else {
            // Others
        }
    }

    return content;
}
```

FAQs

What should I do if "total count of request cloud message exceed max limit" is displayed in the log when I am pulling historical messages?

Currently, the SDK policy is as described below:

1. When `getType` is set to pull cloud historical messages and the number of messages to be pulled is set to `count`, the SDK will pull `count` messages from the cloud.
2. The SDK will filter out invalid messages, such as deleted messages and those irrelevant to the current user.
3. If there are too many invalid historical messages in the cloud, the SDK will perform multiple paged pulls.

To ensure the system stability and robustness, the SDK will perform up to three automatic paged pulls. After this limit is exceeded, the "total count of request cloud message exceed max limit" message will be displayed in the log.

To minimize the impact of such a limit mechanism on the business layer, you can take the following measures to reduce invalid messages:

You can use online messages, that is, set `onlineUserOnly` to `YES/true` when sending messages.

For group messages, you can use targeted group messages to specify the receiver.

What should I do if messages are "lost" during cloud historical message pull?

When `getType` is set to pull cloud historical messages and the number of messages is `count`, the SDK will perform the following operations:

1. Pull `count` messages from the local database.
2. Pull `count` messages from the cloud and filter out invalid messages such as deleted messages. If the number is smaller than `count`, the paged pull will be triggered.
3. Merge local and cloud messages and update information such as the message status.
4. Return `count` messages from the merged message list.

Generally speaking, message loss indicates that too many invalid messages are pulled in step 2, which triggers the limit mechanism in question 1 and leads to an insufficient number of messages actually pulled from the cloud.

We recommend you fix this issue as instructed in question 1. If the issue persists, contact us for assistance.

What should I do if group member information such as the group name card is not updated in real time when historical messages are pulled?

1. When messages are generated, the SDK will update the current group member information such as the group name card and role and store it in the local database.
2. When historical group messages are pulled, the SDK will directly query the group member information when the messages were generated and will not request updates from the backend in real time.

If you want to get the latest group member information, you can use the `getGroupMembersInfo` API (Java / Swift / Objective-C / C++).

What should I do if I get a lag while pulling historical messages?

The SDK has been optimized for message pull. If a lag occurs, you can reduce the number of pulled messages (`count`). If the issue persists, contact us for assistance.

©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript

Last updated：2025-01-07 15:20:40

Feature Overview

The SDK supports message pull by page, sequence, or time range. Historical messages stored in the cloud are subject to the following time limits:
Developer edition: The free storage period is 7 days and cannot be extended.
Standard edition: The free storage period is 7 days and cannot be extended.
Premium edition: The free storage period is 30 days and cannot be extended.

Note

Extending the storage period of historical messages is a Value-Added Service. You can log in to the Chat Console to modify the relevant configuration. For billing details, see Pricing.
Rich media messages (such as images, files, and audios) have the same storage periods as historical messages.

Pulling a Message List

getMessageList

This API is used to pull the list of messages in a specified conversation by page. It needs to be called when the message list is rendered for the first time after the user enters the conversation, or when the user pulls down the list to view more messages.

Note

1. This interface can be used for "pulling historical messages" (excluding AVChatRoom).
2. AVChatRoom does not store historical messages by default. When new users enter the AVChatRoom, they can only see messages sent after they joined. To enhance user engagement and participation in the AVChatRoom, you can update the number of recent messages that new members can view in the Chat console > Application Configuration > Feature Configuration > Group Configuration -> Live Group New Member View Pre-join Message Quantity Configuration. This feature is only available in the Premium edition. New group members can view the latest messages generated within the 24 hours before they joined, up to a maximum of 20 messages.
3. If you want to search for local messages, please use `findMessage` . If you want to search for all cloud messages, please use `searchCloudMessages` (this feature is a Value-Added Service, and you need to purchase the cloud search plugin. Please click to buy).

API

```
chat.getMessageList(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
conversationID	String	Conversation ID, which consists of: `C2C${userID}` (for one-to-one chats) `GROUP${groupID}` (for group chats) `GROUP${topicID}` (topic) `@TIM#SYSTEM` (system notification conversation)
nextReqMessageID	String | undefined	Message ID used for the subsequent paged pull. Leave this field empty for the first pull and enter its the value returned by the `getMessageList` API for the subsequent pull.

Return value

`Promise`

Examples

```
// Pull the message list for the first time when a conversation is opened
let promise = chat.getMessageList({conversationID: 'C2Ctest'});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
  // This parameter must be passed in for the next pull by page.
  const nextReqMessageID = imResponse.data.nextReqMessageID;
  // It indicates whether all messages have been pulled.
  const isCompleted = imResponse.data.isCompleted;
});
```

```
// Pull down the message list to view more messages
let promise = chat.getMessageList({conversationID: 'C2Ctest', nextReqMessageID});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
  // This parameter must be passed in for the next pull by page.
  const nextReqMessageID = imResponse.data.nextReqMessageID;
  // It indicates whether all messages have been pulled.
  const isCompleted = imResponse.data.isCompleted;
});
```

getMessageListHopping

This API is used to pull a message list by specified sequence or time range.

©2013-2025 Tencent Cloud International Pte. Ltd.

API

```
chat.getMessageListHopping(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
conversationID	String	Conversation ID, which consists of: `C2C${userID}` (for one-to-one chats) `GROUP${groupID}` (for group chats) `GROUP${topicID}` (topic).
sequence	Number | undefined	`sequence` of the message starting from which to pull roaming group messages
time	Number | undefined	Server time of the message, starting from which to pull roaming one-to-one messages
direction	Number	Message pull direction, which defaults to `0`. 0: pull in reverse chronological order to get older messages; 1: pull in chronological order to get recent messages
count	Number | undefined	Number of messages to be pulled. It defaults to 15, which is also the maximum value, indicating that up to 15 messages can be pulled and returned at a time.

Return value

`Promise`

Examples

```
// Pull roaming group messages by sequence.
// `direction`: 0: pull in reverse chronological order to get older messages
// 1: pull in chronological order to get recent messages
let promise = chat.getMessageListHopping({
  conversationID: 'GROUPtest',
  sequence: 100,
  count: 15,
  direction: 0
});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// Pull roaming one-to-one messages by time range.
// `direction`: 0: pull in reverse chronological order to get older messages
// 1: pull in chronological order to get recent messages
let promise = chat.getMessageListHopping({
  conversationID: 'C2Ctest',
  time: xxx,
  count: 15,
  direction: 0
});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
});
```

Flutter

Last updated：2025-04-16 14:40:39

Overview

The API for pulling historical messages is in the `TencentImSDKPlugin.v2TIMManager.getMessageManager()` class.
In addition to the support for pulling historical one-to-one and group messages, an advanced API is provided to pull messages by sequence, start point, or time range.
Both local and cloud historical messages can be pulled.
Note:
When historical messages are pulled from the cloud and a network exception is detected, the SDK will return the locally stored historical messages.
Locally stored historical messages are not subject to time limits, but those stored in the cloud are subject to the following time limits:
Free Trial: The free storage period is 7 days and cannot be extended.
Pro edition: The free storage period is 30days and can be extended.
Pro Plus edition: The free storage period is 90days and can be extended.
Enterprise edition: The free storage period is 90days and can be extended.
Note:
It is a value-added service to extend the storage period of historical messages. You can log in to the Chat console to modify relevant configuration. For information about billing, see Value-added Service Pricing.
Rich media messages (such as images, files, and audios) have the same storage periods as historical messages.

Pulling Historical One-to-One Messages

Call the `getC2CHistoryMessageList` API (details) to get historical one-to-one messages.
When the network is normal, the latest cloud data will be pulled; when it is abnormal, the SDK will return the locally stored historical messages.
If you want to pull only local historical messages, see Advanced API.
This API supports pulling by page. For more information, see Pulling by page.
Sample code:

```
// Pull historical one-to-one messages
// Set `lastMsgID` to `null` for the first pull
// `lastMsgID` can be the ID of the last message in the returned message list for t
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
TencentImSDKPlugin.v2TIMManager.getMessageManager().getC2CHistoryMessageList(
        userID: "userId",
        count: 10,
        lastMsgID: null,
);
```

Pulling Historical Group Messages

Call the `getGroupHistoryMessageList` API (details) to get historical group messages.

When the network is normal, the latest cloud data will be pulled; when it is abnormal, the SDK will return the locally stored historical messages.

If you want to pull only local historical messages, see Advanced API.

This API supports pulling by page. For more information, see Pulling by page.

Note:

Only the historical messages of meeting groups (Meeting) can be pulled. For more information on group message limits, see Group System.

This API does not apply to audio-video groups (AVChatRoom), as their messages are not stored on the cloud or in the local database.

Sample code:

```
// Pull historical group messages
// Set `lastMsgID` to `null` for the first pull
// `lastMsgID` can be the ID of the last message in the returned message list for t
TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .getGroupHistoryMessageList(
        groupID: "groupID",
        count: 10,
        lastMsgID: null,
      );
```

Advanced Features

Advanced API

If the ordinary APIs above cannot meet your needs to pull historical messages, you can use the advanced API `getHistoryMessageList` (details).

In addition to pulling historical one-to-one and group messages, this API supports the following advanced features:

Set the source for message pull: pull from the local database or the cloud

©2013-2025 Tencent Cloud International Pte. Ltd.

Specify the sequence for message pull: pull in reverse chronological order or in chronological order

Specify the message type for local pull: text, image, audio, video, file, emoji, group tip, combined, or custom message.

API prototype:

```
// Pull historical one-to-one messages
// Set `lastMsgID` to `null` for the first pull
// `lastMsgID` can be the ID of the last message in the returned message list f
V2TimValueCallback<List<V2TimMessage>> getHistoryMessageListRes =
    await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
        .getHistoryMessageList(
  getType: HistoryMsgGetTypeEnum.V2TIM_GET_LOCAL_OLDER_MSG, // Source and seque
  userID: "userID", // To pull one-to-one messages with a certain user, you nee
  groupID: "groupID", // To pull group messages from a certain group, you need
  count: 10, // Number of messages pulled
  lastMsgID: null, // ID of the first message for pulling
  // This field can be used only for group chats.
  // If `lastMsgSeq` is set as the start point for message, the message will be
  // If both `lastMsg` and `lastMsgSeq` are specified, the SDK will use `lastMs
  // If neither `lastMsg` nor `lastMsgSeq` is specified, the start point for pu
  lastMsgSeq: -1,
  messageTypeList: [], // Used to filter historical message information attribu
);
if (getHistoryMessageListRes.code == 0) {
  // Obtained successfully
}
```

Parameters:

Parameter	Description	Valid for One-to-One Chat	Valid for Group Chat	Required	Remarks
getType	Source and sequence of the message pull, which can be set to **local/cloud** and **reverse chronological order/chronological order** respectively.	YES	YES	YES	When the pull source is set to the cloud, the local message list and cloud message list will be combined and returned. If there is no network connection, the local message list will be returned.
userID	The specified user ID with which to pull historical one-to-one messages	YES	NO	NO	To pull one-to-one messages, you need to specify only the `userID`.

groupID	The specified group ID with which to pull historical group messages	NO	YES	NO	To pull group messages, you need to specify only the `groupID`.
count	Number of messages per pull	YES	YES	YES	We recommend you set it to `20` for iOS/Android clients; otherwise, the pull speed may be affected. For web clients, the maximum value allowed is `15`.
lastMsgID	ID of the last message, indicating the message starting from which to pull historical messages	YES	YES	NO	1. It can be used for both one-to-one and group chats. 2. If it is set as the start point for the message pull, the message will not be included in the returned message list. 3. If it is left empty, the latest message in the conversation will be used as the start point for pull.
lastMsgSeq	`seq` of the last message, indicating the message starting from which to pull historical messages	NO	YES	NO	1. It can be used only for group chats. 2. If it is set as the start point for the message pull, the message will be included in the returned message list. 3. If both lastMsg and lastMsgSeq are specified, the SDK will use lastMsg. 4. If neither lastMsg nor lastMsgSeq is specified, the start point for pull will be determined based on whether the getTimeBegin is set. If yes, the set range will be used as the start point; if no, the latest message will be used as the start point.

Pulling by page

Historical one-to-one message pull, historical group message pull, and pull via the advanced API can all be paged by using `lastMsg` and `count`.

Pulling by page can be implemented by using `lastMsg` and `count`. For the first pull, `lastMsg` can be left empty, which indicates that the last message in the returned message list will be used as the `lastMsg` to pull data of the next page.
When `lastMsg` is left empty, the SDK will return historical messages starting from the latest message by default.
When `lastMsg` is set, it is not included in the returned message list.
The returned message list displays messages in reverse chronological order.
Note:
1. To avoid affecting the speed of pulling historical messages, we recommend you set `count` to `20` for pagination.
2. We recommend you not use `lastMsgSeq` for the subsequent pull, as the corresponding message will be included in the returned message list.

Pulling only local messages

Set `getType` to pull only local messages:
When `getType` is set to `V2TIM_GET_LOCAL_OLDER_MSG`, locally stored messages will be pulled in reverse chronological order.
When `getType` is set to `V2TIM_GET_LOCAL_NEWER_MSG`, locally stored messages will be pulled in chronological order.
The following sample code demonstrates the process of pulling 20 one-to-one messages from the local database in reverse chronological order, starting from the latest message:

```
TencentImSDKPlugin.v2TIMManager.getMessageManager().getHistoryMessageList(
        count: 10,
        getType: HistoryMsgGetTypeEnum.V2TIM_GET_LOCAL_OLDER_MSG,
        userID: "userID",
);
```

Pulling messages after redirecting to the group @ message

After a user receives a group @ message in a group conversation, the user generally needs to click the @ bar to go to the message and pull the neighboring messages for display.
As the group @ message also needs to be displayed, you can set its `sequence` as the `lastMsgSeq` and use the advanced API to pull messages.
The following sample code demonstrates the process of clicking the @ bar, redirecting to the group @ message, and pulling the first 20 messages earlier and later than that message for display:

```
// Get the `sequence` of the group @ message
int atSequence = 1081;

// Pull the group @ message and earlier messages
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
  TencentImSDKPlugin.v2TIMManager.getMessageManager().getHistoryMessageList(
        count: 20, // Pull 20 messages
        getType: HistoryMsgGetTypeEnum.V2TIM_GET_CLOUD_OLDER_MSG, // Pull messages
        lastMsgSeq: atSequence,// Start the pull from the group @ message, which is
        groupID: "groupID" // Pull group messages
      );



// Pull messages later than the group @ message
TencentImSDKPlugin.v2TIMManager.getMessageManager().getHistoryMessageList(
        count: 20, // Pull 20 messages
        getType: HistoryMsgGetTypeEnum.V2TIM_GET_CLOUD_NEWER_MSG, // Pull messages
        lastMsgSeq: atSequence,// Start the pull from the group @ message, which is
        groupID: "groupID" // Pull group messages
      );
```

FAQs

1. What should I do if "total count of request cloud message exceed max limit" is displayed in the log when I am pulling historical messages?

Currently, the SDK policy is as described below:

1. When `getType` is set to pull cloud historical messages and the number of messages to be pulled is set to `x`, the SDK will pull `x` messages from the cloud.

2. The SDK will filter out invalid messages, such as deleted messages and those irrelevant to the current user.

3. If there are too many invalid historical messages in the cloud, the SDK will perform multiple paged pulls.

To ensure the system stability and robustness, the SDK will perform up to three automatic paged pulls. After this limit is exceeded, the "total count of request cloud message exceed max limit" message will be displayed in the log.

To minimize the impact of such a limit mechanism on the business layer, you can take the following measures to reduce invalid messages:

You can use online messages, that is, set `onlineUserOnly` to `YES/true` when sending messages.

For group messages, you can use targeted group messages to specify the receiver.

2. What should I do if messages are "lost" during cloud historical message pull?

When `getType` is set to pull cloud historical messages and the number of messages is `count`, the SDK will perform the following operations:

1. Pull `count` messages from the local database.

©2013-2025 Tencent Cloud International Pte. Ltd.

2. Pull `count` messages from the cloud and filter out invalid messages such as deleted messages. If the number is smaller than `count`, the paged pull will be triggered.
3. Combine local and cloud messages and update information such as the message status.
4. Return `count` messages from the combined message list.
Generally speaking, message loss indicates that too many invalid messages are pulled in step 2, which triggers the limit mechanism in question 1 and leads to an insufficient number of messages actually pulled from the cloud.
We recommend you fix this issue as instructed in question 1.

3. What should I do if group member information such as the group name card is not updated in real time when historical messages are pulled?

When messages are generated, the SDK will update the current group member information such as the group name card and role and store it in the local database.
When historical group messages are pulled, the SDK will directly return the group member information when the messages were generated and will not update it in real time.
If you want to get the latest group member information, you can use the `getGroupMembersInfo` API (details).

4. What should I do if I get a lag while pulling historical messages?

The SDK has been optimized for message pull. If a lag occurs, you can reduce the number of pulled messages (`count`).

©2013-2025 Tencent Cloud International Pte. Ltd.

Unity

Last updated：2025-01-13 16:54:10

Feature Description

The API for pulling historical messages is `MsgGetMsgList` (Details).
It supports pulling historical one-to-one and group messages.
Both local and cloud historical messages can be pulled.
Note:
When historical messages are pulled from the cloud and a network exception is detected, the SDK will return the locally stored historical messages.
Locally stored historical messages are not subject to time limits, but those stored in the cloud are subject to the following time limits:
Free Trial: The free storage period is 7 days and cannot be extended.
Pro edition: The free storage period is 30 days and can be extended.
Pro Plus edition: The free storage period is 90 days and can be extended.
Enterprise edition: The free storage period is 90 days and can be extended.

Note:
Extending the storage period of historical messages is a value-added service. You can log in to the IM console to modify the relevant configuration. For billing details, see Pricing.
Rich media messages (such as images, files, and audios) have the same storage periods as historical messages.

Pulling Historical One-to-One Messages

Call the `MsgGetMsgList` API (Details) to get historical one-to-one messages.
When the network is normal, the latest cloud data will be pulled; when it is abnormal, the SDK will return the locally stored historical messages.
This API supports pulling by page. For more information, see Pulling by page.
Sample code:

```
// Pull historical one-to-one messages
// Set `msg_getmsglist_param_last_msg` to `null` for the first pull
// `msg_getmsglist_param_last_msg` can be the last message in the returned message
var get_message_list_param = new MsgGetMsgListParam
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    {
      msg_getmsglist_param_last_msg = LastMessage
    };
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
  // Process the callback logic
});
```

Pulling Historical Group Messages

Call the `MsgGetMsgList` API (Details) to get historical group messages.

When the network is normal, the latest cloud data will be pulled; when it is abnormal, the SDK will return the locally stored historical messages.

This API supports pulling by page. For more information, see Pulling by page.

Caution:

Only the historical messages of meeting groups (Meeting) can be pulled. For more information on group message limits, see Group System.

This API does not apply to audio-video groups (AVChatRoom), as their messages are not stored on the cloud roaming server or in the local database.

Sample code:

```
// Pull historical group messages
// Set `msg_getmsglist_param_last_msg` to `null` for the first pull
// `msg_getmsglist_param_last_msg` can be the last message in the returned message
var get_message_list_param = new MsgGetMsgListParam
    {
      msg_getmsglist_param_last_msg = LastMessage
    };
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_Group, get
  // Process the callback logic
});
```

Pulling by page

Historical one-to-one message pull and historical group message pull can all be paged by using `msg_getmsglist_param_last_msg` and `msg_getmsglist_param_count`.

Pulling by page can be implemented by using `msg_getmsglist_param_last_msg` and `msg_getmsglist_param_count`. For the first pull, `msg_getmsglist_param_last_msg` can be left

©2013-2025 Tencent Cloud International Pte. Ltd.

empty, which indicates that the last message in the returned message list will be used as the `msg_getmsglist_param_last_msg` to pull data of the next page.
When `msg_getmsglist_param_last_msg` is left empty, the SDK will return historical messages starting from the latest message by default.
When `msg_getmsglist_param_last_msg` is set, it is not included in the returned message list.
The returned message list displays messages in reverse chronological order.
Note:
1. To avoid affecting the speed of pulling historical messages, we recommend you set `msg_getmsglist_param_count` to `20` for pagination.
2. We recommend you not use `msg_getmsglist_param_last_msg_seq` for the subsequent pull, as the corresponding message will be included in the returned message list.

Pulling messages after redirecting to the group @ message

After a user receives a group @ message in a group conversation, the user generally needs to click the @ bar to go to the message and pull the neighboring messages for display.
As the group @ message also needs to be displayed, you can set its `sequence` as the `msg_getmsglist_param_last_msg_seq` and use the `MsgGetMsgList` to pull messages.

FAQs

1. What should I do if "total msg_getmsglist_param_count of request cloud message exceed max limit" is displayed in the log when I am pulling historical messages?

Currently, the SDK policy is as described below:
1. When `getType` is set to pull cloud historical messages and the number of messages to be pulled is set to `x`, the SDK will pull `x` messages from the cloud.
2. The SDK will filter out invalid messages, such as deleted messages and those irrelevant to the current user.
3. If there are too many invalid historical messages in the cloud, the SDK will perform multiple paged pulls.
To ensure the system stability and robustness, the SDK will perform up to three automatic paged pulls. After this limit is exceeded, the "total msg_getmsglist_param_count of request cloud message exceed max limit" message will be displayed in the log.
To minimize the impact of such a limit mechanism on the business layer, you can take the following measures to reduce invalid messages:
You can use online messages, that is, set `message_is_online_msg` to `YES/true` when sending messages.

©2013-2025 Tencent Cloud International Pte. Ltd.

For group messages, you can use targeted group messages to specify the receiver.

2. What should I do if messages are "lost" during cloud historical message pull?

When `getType` is set to pull cloud historical messages and the number of messages is `msg_getmsglist_param_count`, the SDK will perform the following operations:
1. Pull `msg_getmsglist_param_count` messages from the local database.
2. Pull `msg_getmsglist_param_count` messages from the cloud and filter out invalid messages such as deleted messages. If the number is smaller than `msg_getmsglist_param_count`, the paged pull will be triggered.
3. Merge local and cloud messages and update information such as the message status.
4. Return `msg_getmsglist_param_count` messages from the merged message list.
Generally speaking, message loss indicates that too many invalid messages are pulled in step 2, which triggers the limit mechanism in question 1 and leads to an insufficient number of messages actually pulled from the cloud.
We recommend you fix this issue as instructed in question 1. If the issue persists, contact us for assistance.

3. What should I do if group member information such as the group name card is not updated in real time when historical messages are pulled?

When messages are generated, the SDK will update the current group member information such as the group name card and role and store it in the local database.
When historical group messages are pulled, the SDK will directly return the group member information when the messages were generated and will not update it in real time.
If you want to get the latest group member information, you can use the `GroupGetMemberInfoList` API (Details).

4. What should I do if I get a lag while pulling historical messages?

The SDK has been optimized for message pull. If a lag occurs, you can reduce the number of pulled messages (`msg_getmsglist_param_count`). If the issue persists, contact us for assistance.

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

Last updated：2025-03-20 16:23:27

Feature Overview

The SDK supports message pull by page, sequence, or time range. Historical messages stored in the cloud are subject to the following time limits:
Developer edition: The free storage period is 7 days and cannot be extended.
Standard edition: The free storage period is 7 days and cannot be extended.
Pro edition :The free storage period is 30 days and cannot be extended.
Pro Plus edition :The free storage period is 60 days and cannot be extended.
Enterprise edition: The free storage period is 90 days and cannot be extended.
Note
Extending the storage period of historical messages is a Value-Added Service. You can log in to the Chat Console to modify the relevant configuration. For billing details, see Pricing.
Rich media messages (such as images, files, and audios) have the same storage periods as historical messages.

Pulling a Message List

getMessageList

This API is used to pull the list of messages in a specified conversation by page. It needs to be called when the message list is rendered for the first time after the user enters the conversation, or when the user pulls down the list to view more messages.
Note
1. This interface can be used for "pulling historical messages" (excluding AVChatRoom).
2. AVChatRoom does not store historical messages by default. When new users enter the AVChatRoom, they can only see messages sent after they joined. To enhance user engagement and participation in the AVChatRoom, you can update the number of recent messages that new members can view in the Chat console > Application Configuration > Feature Configuration > Group Configuration -> Live Group New Member View Pre-join Message Quantity Configuration. This feature is only available in the Pro edition 、 Pro Plus edition or Enterprise edition. New group members can view the latest messages generated within the 24 hours before they joined, up to a maximum of 20 messages.
3. If you want to search for local messages, please use `findMessage` . If you want to search for all cloud messages, please use `searchCloudMessages` (this feature is a Value-Added Service, and you need to purchase the cloud search plugin. Please click to buy).

©2013-2025 Tencent Cloud International Pte. Ltd.

API

```
chat.getMessageList(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
conversationID	String	Conversation ID, which consists of: `C2C${userID}` (for one-to-one chats) `GROUP${groupID}` (for group chats) `GROUP${topicID}` (topic) `@TIM#SYSTEM` (system notification conversation)
nextReqMessageID	String | undefined	Message ID used for the subsequent paged pull. Leave this field empty for the first pull and enter its the value returned by the `getMessageList` API for the subsequent pull.

Return value

`Promise`

Examples

```
// Pull the message list for the first time when a conversation is opened
let promise = chat.getMessageList({conversationID: 'C2Ctest'});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
  // This parameter must be passed in for the next pull by page.
  const nextReqMessageID = imResponse.data.nextReqMessageID;
  // It indicates whether all messages have been pulled.
  const isCompleted = imResponse.data.isCompleted;
});
```

```
// Pull down the message list to view more messages
let promise = chat.getMessageList({conversationID: 'C2Ctest', nextReqMessageID});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
  // This parameter must be passed in for the next pull by page.
  const nextReqMessageID = imResponse.data.nextReqMessageID;
  // It indicates whether all messages have been pulled.
  const isCompleted = imResponse.data.isCompleted;
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

getMessageListHopping

This API is used to pull a message list by specified sequence or time range.

API

```
chat.getMessageListHopping(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
conversationID	String	Conversation ID, which consists of: `C2C${userID}` (for one-to-one chats) `GROUP${groupID}` (for group chats) `GROUP${topicID}` (topic).
sequence	Number | undefined	`sequence` of the message starting from which to pull roaming group messages
time	Number | undefined	Server time of the message, starting from which to pull roaming one-to-one messages
direction	Number	Message pull direction, which defaults to `0`. 0: pull in reverse chronological order to get older messages; 1: pull in chronological order to get recent messages
count	Number | undefined	Number of messages to be pulled. It defaults to 15, which is also the maximum value, indicating that up to 15 messages can be pulled and returned at a time.

Return value

`Promise`

Examples

```
// Pull roaming group messages by sequence.
// `direction`: 0: pull in reverse chronological order to get older messages
// 1: pull in chronological order to get recent messages
let promise = chat.getMessageListHopping({
  conversationID: 'GROUPtest',
  sequence: 100,
  count: 15,
  direction: 0
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
});
```

```
// Pull roaming one-to-one messages by time range.
// `direction`: 0: pull in reverse chronological order to get older messages
// 1: pull in chronological order to get recent messages
let promise = chat.getMessageListHopping({
  conversationID: 'C2Ctest',
  time: xxx,
  count: 15,
  direction: 0
});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // Message list
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Forwarding Message Android&iOS&Windows&Mac

Last updated：2025-01-06 17:39:51

Feature Description

You can combine and forward messages in the following steps:

1. Create a combined message based on the list of original messages.
2. Send the combined message to the receiver.
3. The receiver receives the combined message and parses the list of original messages.

The title and digest are needed to display the combined message, as shown below:

Combine and Forward	Display of Combined Message

©2013-2025 Tencent Cloud International Pte. Ltd.

Combining and Forwarding Messages

Creating and sending a combined message

A combined message can be created by setting the message list along with the combined message title and digest. The process is as follows:

1. Call the `createMergerMessage` API (Java / Swift / Objective-C / C++) to create a combined message. The list of original messages as well as the combined message title and digest also need to be set.

Attribute	Description	Remarks
messageList	List of original messages	List of original messages to be combined and forwarded
title	Title	Title of the combined message, such as "Chat History of xixiyah and Hello" as shown above
abstractList	Digest list	Digest list of the combined message as shown above. The original message digests need to be displayed for the combined message, which will be unfolded after the user clicks the cell.
compatibleText	Compatibility text message	If the early SDK version does not support the combined message, the user will receive a text message with the content `compatibleText` by default.

Sample code:

Java

Swift

Objective-C

C++

```
// List of messages to be forwarded, which can contain combined messages but not gr
List<V2TIMMessage> msgs = new ArrayList<>();
msgs.add(message1);
msgs.add(message2);
// Title of the combined message
String title = "Chat History of vinson and lynx";
// Digest list of the combined message
List<String> abstactList = new ArrayList<>();
msgs.add("abstract1");
```

```
msgs.add("abstract2");
msgs.add("abstract3");
// Compatibility text of the combined message. If the early SDK version does not su
String compatibleText = "Upgrade to the latest version to check the merged message"
// Create a combined message
V2TIMMessage mergeMessage = V2TIMManager.getMessageManager().createMergerMessage(ms
```

```
// List of messages to be forwarded, which can contain combined messages but not gr
let msgs: [V2TIMMessage] = [message1, message2 /* ... */]
// Title of the combined message
let title = "Chat History of vinson and lynx"
// Digest list of the combined message
let abstractList = ["abstract1", "abstract2", "abstract3"]
// Compatibility text of the combined message. If the early SDK version does not su
let compatibleText = "Upgrade to the latest version to check the merged message"
// Create a combined message
let mergeMessage = V2TIMManager.sharedInstance().createMergerMessage(messageList: m
}
```

```
// List of messages to be forwarded, which can contain combined messages but not gr
NSArray *msgs = @[message1,message2...];
// Title of the combined message
NSString *title = @"Chat History of vinson and lynx";
// Digest list of the combined message
NSArray *abstactList = @[@"abstract1",@"abstract2",@"abstract3"];
// Compatibility text of the combined message. If the SDK on an early version does
NSString *compatibleText = @"Upgrade to the latest version to check the merged mess
// Create a combined message
V2TIMMessage *mergeMessage = [[V2TIMManager sharedInstance] createMergerMessage:msg
abstractList:abstactList compatibleText:compatibleText];
```

```
// List of messages to be forwarded, which can contain combined messages but not gr
V2TIMMessageVector messageList;
messageList.PushBack(message1);
messageList.PushBack(message2);
// Title of the combined message
V2TIMString title = "Chat History of vinson and lynx";
// Digest list of the combined message
V2TIMStringVector abstractList;
abstractList.PushBack("abstract1");
abstractList.PushBack("abstract2");
abstractList.PushBack("abstract3");
// Compatibility text of the combined message. If the early SDK version does not su
// the user will receive a text message with the content `compatibleText` by defaul
V2TIMString compatibleText = "Upgrade to the latest version to check the merged mes
```

```
// Create a combined message
V2TIMMessage mergerMessage = V2TIMManager::GetInstance()->GetMessageManager()->Crea
messageList, title, abstractList, compatibleText);
```

2. Call the `sendMessage` API (Java / Swift / Objective-C / C++) to send the combined message.

Sample code:

Java

Swift

Objective-C

C++

```
// Send the combined message to the user `denny`
V2TIMManager.getMessageManager().sendMessage(mergeMessage, "denny", null, V2TIMMess
  @Override
  public void onProgress(int progress) {}

  @Override
  public void onSuccess(V2TIMMessage v2TIMMessage) {}

  @Override
  public void onError(int code, String desc) {}
  })
```

```
// Send the combined message to the user `denny`
V2TIMManager.shared.sendMessage(message: msg, receiver: "denny", groupID: nil, prio
                offlinePushInfo: nil) { progress in
    //
}succ: {
    print("success")

}fail: { code, desc in
    print("failure, code: \\(code), desc: \\(desc)")
}
```

```
// Send the combined message to the user `denny`
[[V2TIMManager sharedInstance] sendMessage:mergeMessage receiver:@"denny"  groupID:
 priority:V2TIM_PRIORITY_NORMAL  onlineUserOnly:NO  offlinePushInfo:nil progress:ni
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;
```

```
    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

auto callback = new SendCallback{};
callback->SetCallback([=](const V2TIMMessage& message) { delete callback; },
                      [=](int error_code, const V2TIMString& error_message) { delet
                      [=](uint32_t progress) {});

V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    mergerMessage, "denny", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMAL, false,
```

Receiving a combined message

Adding a listener

The receiver can call `addAdvancedMsgListener` (Java / Swift / Objective-C / C++) to add the advanced message listener.

We recommend it be called early, such as after the chat page is initialized, to ensure timely message receiving in the application.

Sample code:

Java

Swift

Objective-C

C++

```
V2TIMManager.getMessageManager().addAdvancedMsgListener(advancedMsgListener);
```

```
// self implements V2TIMAdvancedMsgListener protocol
V2TIMManager.shared.addAdvancedMsgListener(listener: self)
```

```
// `self` is id<V2TIMAdvancedMsgListener>
[[V2TIMManager sharedInstance] addAdvancedMsgListener:self];
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
    // Members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Parsing a message

After the listener is added, the receiver will receive the combined message `V2TIMMessage` in `onRecvNewMessage`.

You can use the combined message element `V2TIMMergerElem` (Java / Swift / Objective-C / C++) to get the `title` and `abstractList` for UI display.

Then, when the user clicks the combined message, you can call the `downloadMergerMessage` API (Java / Swift / Objective-C / C++) to download the combined message list for UI display.

Sample code:

Java

Swift

Objective-C

C++

```
@Override
public void onRecvNewMessage(V2TIMMessage msg) {
```

```
  if (msg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_MERGER) {
    // Get the elements of the combined message
    V2TIMMergerElem mergerElem = msg.getMergerElem();
    // Get the `title`
    String title = mergerElem.getTitle();
    // Get the digest list
    List<String> abstractList = mergerElem.getAbstractList();
    // Download the combined message list when the user clicks the combined message
    mergerElem.downloadMergerMessage(new V2TIMValueCallback<List<V2TIMMessage>>() {
      @Override
      public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        // Downloaded successfully. `v2TIMMessages` is the combined message list.
        for (V2TIMMessage subMsg : v2TIMMessages) {
        // If the combined message list still contains a combined message, continue
        if (subMsg.getElemType() == V2TIMMessage.V2TIM_ELEM_TYPE_MERGER) {
          V2TIMMergerElem mergerElem = subMsg.getMergerElem();
          // Get the `title`
          String title = mergerElem.getTitle();
          // Get the digest list
          List<String> abstractList = mergerElem.getAbstractList();
          // Download the combined message list when the user clicks the combined m
          ......
        }
      }
    }

    @Override
    public void onError(int code, String desc) {
    // Download failed
    }
  });
}
```

```
func onRecvNewMessage(_ msg: V2TIMMessage) {
    if msg.elemType == .V2TIM_ELEM_TYPE_MERGER {
        // Get the elements of the combined message
        let mergerElem = msg.mergerElem
        // Get the `title`
        let title = mergerElem?.title
        // Get the digest list
        let abstractList = mergerElem?.abstractList

        // Download the combined message list when the user clicks the combined mes
        mergerElem?.downloadMergerMessage { msgs in
                    // Downloaded successfully. `v2TIMMessages` is the combined mes
            for subMsg in msgs {
```

```
                    // If the combined message list still contains a combined messa
                if subMsg.elemType == .V2TIM_ELEM_TYPE_MERGER {
                    let subMergerElem = subMsg.mergerElem
                    // Get the `title`
                    let subTitle = subMergerElem?.title
                    // Get the digest list
                    let subAbstractList = subMergerElem?.abstractList

                    // Download the combined message list when the user clicks the
                    subMergerElem?.downloadMergerMessage(succ: { msgs in

                    }, fail: { code, desc in

                    })
                }
            }
        } fail: { code, desc in
            // Download failed
        }
    }
}
```

```
- (void)onRecvNewMessage:(V2TIMMessage *)msg {
    if (msg.elemType == V2TIM_ELEM_TYPE_MERGER) {
        // Get the elements of the combined message
        V2TIMMergerElem *mergerElem = msg.mergerElem;
        // Get the `title`
        NSString *title = mergerElem.title;
        // Get the digest list
        NSArray *abstractList = mergerElem.abstractList;
        // Download the combined message list when the user clicks the combined mes
        [msg.mergerElem downloadMergerMessage:^(NSArray<V2TIMMessage *> *msgs) {
            // Downloaded successfully. `msgs` is the combined message list.
            for (V2TIMMessage *subMsg in msgs) {
                // If the combined message list still contains a combined message,
                if (subMsg.elemType == V2TIM_ELEM_TYPE_MERGER) {
                    V2TIMMergerElem *mergerElem = subMsg.mergerElem;
                    // Get the `title`
                    NSString *title = mergerElem.title;
                    // Get the digest list
                    NSArray *abstractList = mergerElem.abstractList;
                    // Download the combined message list when the user clicks the
                    [msg.mergerElem downloadMergerMessage:nil fail:nil];
                }
            }
        } fail:^(int code, NSString *desc) {
```

```
            // Download failed
        }];
    }
}
```

```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
        if (success_callback_) {
            success_callback_(value);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        if (message.elemList.Size() == 1) {
            V2TIMElem* elem = message.elemList[0];
            if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_MERGER) {
                // Get the elements of the combined message
                auto mergerElem = static_cast<V2TIMMergerElem*>(elem);
                // Get the `title`
                V2TIMString title = mergerElem->title;
                /// Get the digest list
```

```
                V2TIMStringVector abstractList = mergerElem->abstractList;

                // Download the combined message list when the user clicks the comb
                auto callback = new ValueCallback<V2TIMMessageVector>{};
                callback->SetCallback(
                    [=](const V2TIMMessageVector& messageList) {
                        // Downloaded successfully. `v2TIMMessages` is the combined
                        for (size_t i = 0; i < messageList.Size(); ++i) {
                            const V2TIMMessage& message = messageList[i];
                            if (message.elemList.Size() == 1) {
                                V2TIMElem* elem = message.elemList[0];
                                // If the combined message list still contains a co
                                if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYP
                                    // ...
                                }
                            }
                        }
                        delete callback;
                    },
                    [=](int error_code, const V2TIMString& error_message) { delete

                mergerElem->DownloadMergerMessage(callback);
            }
        }
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

Removing a listener

To stop receiving messages, the receiver can call `removeAdvancedMsgListener` (Java / Swift / Objective-C / C++) to remove the advanced message listener.

Sample code:

Java

Swift

Objective-C

C++

```
V2TIMManager.getMessageManager().removeAdvancedMsgListener(advancedMsgListener);
```

```
V2TIMManager.shared.removeAdvancedMsgListener(listener: self)
```

```
// `self` is id<V2TIMAdvancedMsgListener>
[[V2TIMManager sharedInstance] removeAdvancedMsgListener:self];
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
    // Members ...
};

// `advancedMsgListener` is the instance of AdvancedMsgListener
V2TIMManager::GetInstance()->GetMessageManager()->RemoveAdvancedMsgListener(&advanc
```

Forwarding Messages One by One

To forward a single message, create a message identical to the original message through the `createForwardMessage` API (Java / Swift / Objective-C / C++) first, and then call the `sendMessage` API (Java / Swift / Objective-C / C++) to send the message.

Sample code:

Java

Swift

Objective-C

C++

```
// Create a message with the same elements as the original message
V2TIMMessage forwardMessage = V2TIMManager.getMessageManager().createForwardMessage
// Send the message to the user `denny`
V2TIMManager.getMessageManager().sendMessage(forwardMessage, "denny", null, V2TIMMe
  @Override
  public void onProgress(int progress) {
  }

  @Override
  public void onSuccess(V2TIMMessage message) {
  }

  @Override
  public void onError(int code, String desc) {
  }
});
```

```
// Create a message with the same elements as the original message
if  let originMsg = sendedMsgList.first,
    let msg = V2TIMManager.shared.createForwardMessage(message: originMsg) {
// Send the message to the user `denny`
    _ = V2TIMManager.shared.sendMessage(message: msg, receiver: "denny", groupID: n

    } succ: {
    } fail: { code, desc in

    }
}
```

```
// Create a message with the same elements as the original message
V2TIMMessage *forwardMessage = [[V2TIMManager sharedInstance] createForwardMessage:
// Send the message to the user `denny`
[[V2TIMManager sharedInstance] sendMessage:forwardMessage receiver:@"denny"  groupI
 priority:V2TIM_PRIORITY_NORMAL  onlineUserOnly:NO  offlinePushInfo:nil progress:ni
```

```
class SendCallback final : public V2TIMSendCallback {
public:
    using SuccessCallback = std::function<void(const V2TIMMessage&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;
    using ProgressCallback = std::function<void(uint32_t)>;

    SendCallback() = default;
    ~SendCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
                     ProgressCallback progress_callback) {
        success_callback_ = std::move(success_callback);
        error_callback_ = std::move(error_callback);
        progress_callback_ = std::move(progress_callback);
    }

    void OnSuccess(const V2TIMMessage& message) override {
        if (success_callback_) {
            success_callback_(message);
        }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
        if (error_callback_) {
            error_callback_(error_code, error_message);
        }
    }
    void OnProgress(uint32_t progress) override {
```

```
        if (progress_callback_) {
            progress_callback_(progress);
        }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
    ProgressCallback progress_callback_;
};

// Create a message with the same elements as the original message
V2TIMMessage forwardMessage = V2TIMManager::GetInstance()->GetMessageManager()->Cre

auto callback = new SendCallback{};
callback->SetCallback([=](const V2TIMMessage& message) { delete callback; },
                      [=](int error_code, const V2TIMString& error_message) { delet
                      [=](uint32_t progress) {});

// Send the message to the user `denny`
V2TIMManager::GetInstance()->GetMessageManager()->SendMessage(
    forwardMessage, "denny", {}, V2TIMMessagePriority::V2TIM_PRIORITY_NORMAL, false
```

JavaScript

Last updated：2025-01-07 15:20:40

Feature Overview

You can merge and forward messages in the following steps:

1. Create a merged message based on the list of original messages.
2. Send the merged message to the receiver.
3. The receiver receives the merged message and parses the list of original messages.

UI Display

Creating a Merged Message

This API is used to create a merged message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a merged message.

Note:

1. Unable to merge messages that failed to be sent. If the message list contains a message that failed to be sent, the API will report an error.

API

```
chat.createMergerMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
messageList	Array	Merged message list
title	String	Title of merged messages, for example, "Chat History of the Talent Center in the Greater Bay Area"
abstractList	String	Digest list. You can set digest information in different formats for different message types, for example, in the `sender:text` format for a text message, in the `sender:[image]` format for an image message, or in the `sender:[file]` format for a file message.
compatibleText	String	Compatibility text. If the early SDK version does not support the merged message, the user will receive a text message with the content `${compatibleText}` by default. This field is required.

©2013-2025 Tencent Cloud International Pte. Ltd.

Return value

`Message`

Examples

```
// 1. Forward group messages to a one-to-one conversation.
// `message1`, `message2`, and `message3` are group messages.
let mergerMessage = chat.createMergerMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    messageList: [message1, message2, message3],
    title: 'Chat History of the Talent Center in the Greater Bay Area',
    abstractList: ['allen: 666', 'iris: [Image]', 'linda: [File]'],
    compatibleText: 'Upgrade your Chat SDK to v2.10.1 or later to view this message
  },
  // cloudCustomData: 'your cloud custom data'
});

// 2. Send the message.
let promise = chat.sendMessage(mergerMessage);
promise.then(function(imResponse) {
  // Message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Downloading a Merged Message

This API is used to download a merged message. When the merged message sent by the sender is large in size, the SDK will store it in the cloud, and the message receiver needs to download it from the cloud before viewing it.

API

```
chat.downloadMergerMessage(message);
```

Parameters

Name	Type	Description
message	Message	Message instance

©2013-2025 Tencent Cloud International Pte. Ltd.

Return value

`Promise`

Examples

```
// If `downloadKey` exists, the received merged message is stored in the cloud
// and needs to be downloaded first.
if (message.type === TencentCloudChat.TYPES.MSG_MERGER && message.payload.downloadK
  let promise = chat.downloadMergerMessage(message);
  promise.then(function(imResponse) {
    // After the download is successful
    // the SDK will update information such as `message.payload.messageList`.
    console.log(imResponse.data);
  }).catch(function(imError) {
    // Download failed
    console.warn('downloadMergerMessage error:', imError);
  });
}
```

Forwarding Messages One by One

To forward a single message, create a message identical to the original message through the `createForwardMessage` API first, and then call the `sendMessage` API to send the message.

API

```
chat.createForwardMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver

conversationType	String	Conversation type. Valid values:
		`TencentCloudChat.TYPES.CONV_C2C`
		(one-to-one conversation)
		`TencentCloudChat.TYPES.CONV_GROUP`
		(group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority

		messages first. Supported enumerated values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH` `TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL` (default) `TencentCloudChat.TYPES.MSG_PRIORITY_LOW` `TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`
payload	Message	Message instance
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

Return value

`Message`

Examples

```
let forwardMessage = chat.createForwardMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: message, // Message instance for the received or sent message
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send the message.
let promise = chat.sendMessage(forwardMessage);
promise.then(function(imResponse) {
  // Message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Flutter

Last updated：2025-04-16 14:40:39

Feature Description

You can merge and forward messages as provided by WeChat in the following steps:

1. Create a merged message based on the list of original messages.
2. Send the merged message to the receiver.
3. The receiver receives the merged message and parses the list of original messages.

The title and digest are needed to display the merged message, as shown below:

Merge and Forward	Display of Merged Message	Click Merged Message to Download Message List for Display

Merging and Forwarding Messages

Creating and sending a merged message

A merged message can be created by setting the message list along with the merged message title and digest. The process is as follows:

1. Call the `createMergerMessage` API (Details) to create a merged message. The list of original messages as well as the merged message title and digest also need to be set.

Attribute	Description	Remarks
msgIDList	List of IDs of original messages	List of IDs of original messages to be merged and forwarded
title	Title	Title of the merged message, such as "Chat History of xixiyah and Hello" as shown above
abstractList	Digest list	Digest list of the merged message as shown above. The original message

©2013-2025 Tencent Cloud International Pte. Ltd.

		digests need to be displayed for the merged message, which will be unfolded after the user clicks the cell.
compatibleText	Compatibility text message	If the early SDK version does not support the merged message, the user will receive a text message with the content `compatibleText` by default.

Sample code for creating and sending a merged message:

```
// List of messages to be forwarded, which can contain merged messages but not grou
V2TimValueCallback<V2TimMsgCreateInfoResult> createMergerMessageResult =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createMergerMessage(
            msgIDList: ["msgid1", "msgid2"],
            title: "Chat History of user1 and user2", // Title of the merged messag
            abstractList: ["user1:hello", "user2:hello"], // Digest list of the mer
            compatibleText: "The current version does not support the message", //
          );
  if (createMergerMessageResult.code == 0) {
    TencentImSDKPlugin.v2TIMManager.getMessageManager().sendMessage(
          id: createMergerMessageResult.data.id,
          receiver: "",
          groupID: "",
        );
  }
```

Receiving a merged message

Adding a listener

The receiver calls `addAdvancedMsgListener` (Details) to add the advanced message listener.

We recommend it be called early, such as after the chat page is initialized, to ensure timely message receiving in the application.

Sample code:

```
TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
      .addAdvancedMsgListener(listener: listener);
```

Parsing a message

After the listener is added, the receiver will receive the merged message `V2TimMessage` in `onRecvNewMessage`.

You can use the merged message element `V2TimMergerElem` (Details) to get the `title` and `abstractList` for UI display.
Then, when the user clicks the merged message, you can call the `downloadMergerMessage` API (Details) to download the merged message list for UI display.
Sample code:

```
if(message.elemType == MessageElemType.V2TIM_ELEM_TYPE_MERGER){
        message.mergerElem.abstractList;
        message.mergerElem.isLayersOverLimit;
        message.mergerElem.title;
        V2TimValueCallback<List<V2TimMessage>> download = await TencentImSDKPlugin.
        if(download.code == 0){
         List<V2TimMessage> messageList = download.data;
        }
}
```

Forwarding Messages One by One

To forward a single message, create a message identical to the original message through the `createForwardMessage` API (Details) first, and then call the `sendMessage` API (Details) to send the message.
Sample code:

```
// Create a message with the same elements as the original message
V2TimValueCallback<V2TimMsgCreateInfoResult>  createForwardMessageRes = await Tence
// Send the message to the user `denny`
  if(createForwardMessageRes.code == 0){
    TencentImSDKPlugin.v2TIMManager.getMessageManager().sendMessage(id: createForwa
  }
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Unity

Last updated：2024-01-31 16:12:15

Overview

You can implement the feature of combining and forwarding messages in the following steps:
1. Create a combined message based on the list of original messages.
2. Send the combined message to the receiver.
3. The receiver receives the combined message and parses the list of original messages.

Combined Message

Creating and sending a combined message

A combined message can be created by setting the message list along with the combined message title and digest. The process is as follows:
1. Create a combined message by setting the list of original messages as well as the combined message title and digest.

Attribute	Definition	Description
merge_elem_message_array	List of original messages	List of original messages to be combined and forwarded
merge_elem_title	Title	Title of the combined message, such as "Chat History of xixiyah and Hello" as shown above
merge_elem_abstract_array	Digest list	Digest list of the combined message as shown above. The original message digests need to be displayed for the combined message, which will be unfolded after the user clicks the cell.
merge_elem_compatible_text	Compatibility text message	If the early SDK versions do not support the combined message, the user will receive a text message with the content `merge_elem_compatible_text` by default.

2. Sample code for creating and sending a combined message:

©2013-2025 Tencent Cloud International Pte. Ltd.

```
// List of messages to be forwarded, which can contain combined messages but
not group tips
var message = new Message
{
   message_conv_id = conv_id,
   message_conv_type = TIMConvType.kTIMConv_Group,
   message_elem_array = new List<Elem>{
    new Elem
   {
     elem_type = TIMElemType.kTIMElem_Merge,
     merge_elem_title = "Chat History of user1 and user2", // Title of the
combined message
     merge_elem_message_array = new List<Message>
     {
      message1,
      message2
     },
     merge_elem_abstract_array = new List<string>
     {
      "user1:hello", "user2:hello" // Digest list of the combined message
     },
     merge_elem_compatible_text = "The current version does not support the
message" // Compatibility text of the combined message. If the early SDK
version does not support the combined message, the user will receive a text
message with the content `compatibleText` by default.
   }},
};
StringBuilder messageId = new StringBuilder(128);
TIMResult res = TencentIMSDK.MsgSendMessage(conv_id,
TIMConvType.kTIMConv_Group, message, messageId, (int code, string desc, string
json_param, string user_data)=>{
 // Async message sending result
});
```

Receiving a combined message

Adding a listener

The receiver calls the `AddRecvNewMsgCallback` API (details) to add a message listener.

We recommend it be called early, such as after the chat page is initialized, to ensure timely message receiving in the application.

Sample code:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
```

©2013-2025 Tencent Cloud International Pte. Ltd.

```
    foreach(Message message in messages)
    {
      foreach (Elem elem in message.message_elem_array)
      {
        // There is a next message
        if (elem.elem_type == TIMElemType.kTIMElem_Merge)
        {
        }
      }
    }
})
```

Parsing a message

After the listener is added, the receiver will receive the combined message `Message` in `RecvNewMsgCallback`.

You can use the combined message element to get the `merge_elem_title` and `merge_elem_abstract_array` for UI display.

Then, when the user clicks the combined message, you can call the `MsgDownloadMergerMessage` API (details) to download the combined message list for UI display.

Sample code:

```
if(elem.TIMElemType == TIMElemType.kTIMElem_Merge){
    elem.merge_elem_abstract_array;
    elem.merge_elem_layer_over_limit;
    elem.merge_elem_title;
    TIMResult res = TencentIMSDK.MsgDownloadMergerMessage(message, (int code, string
     // Process the async logic
    });
}
```

Forwarding Messages One by One

To forward a single message, create a message identical to the original message first, and then call the `MsgSendMessage` API (details) to send the message.

©2013-2025 Tencent Cloud International Pte. Ltd.

React Native

Last updated：2025-01-07 15:20:40

Feature Overview

You can merge and forward messages in the following steps:

1. Create a merged message based on the list of original messages.
2. Send the merged message to the receiver.
3. The receiver receives the merged message and parses the list of original messages.

UI Display

Merge and Forward	Display of Merged Message

Creating a Merged Message

This API is used to create a merged message. It returns a message instance, which can be sent by calling the `sendMessage` API when you need to send a merged message.

Note:

1. Unable to merge messages that failed to be sent. If the message list contains a message that failed to be sent, the API will report an error.

API

```
chat.createMergerMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)

priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values:
		`TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`
		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL (default)`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Object	Message content container
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

The `payload` is as described below:

Name	Type	Description
messageList	Array	Merged message list
title	String	Title of merged messages, for example, "Chat History of the Talent Center in the Greater Bay Area"
abstractList	String	Digest list. You can set digest information in different formats for different message types, for example, in the `sender:text` format for a text message, in the `sender:[image]` format for an image message, or in the `sender:[file]` format for a file message.
compatibleText	String	Compatibility text. If the early SDK version does not support the merged message, the user will receive a text message with the content `${compatibleText}` by default. This field is required.

Return value

`Message`

Examples

```
// 1. Forward group messages to a one-to-one conversation.
// `message1`, `message2`, and `message3` are group messages.
let mergerMessage = chat.createMergerMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: {
    messageList: [message1, message2, message3],
    title: 'Chat History of the Talent Center in the Greater Bay Area',
    abstractList: ['allen: 666', 'iris: [Image]', 'linda: [File]'],
    compatibleText: 'Upgrade your Chat SDK to v2.10.1 or later to view this message
  },
  // cloudCustomData: 'your cloud custom data'
});

// 2. Send the message.
let promise = chat.sendMessage(mergerMessage);
promise.then(function(imResponse) {
  // Message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

Downloading a Merged Message

This API is used to download a merged message. When the merged message sent by the sender is large in size, the SDK will store it in the cloud, and the message receiver needs to download it from the cloud before viewing it.

API

```
chat.downloadMergerMessage(message);
```

Parameters

Name	Type	Description
message	Message	Message instance

Return value

`Promise`

©2013-2025 Tencent Cloud International Pte. Ltd.

Examples

```
// If `downloadKey` exists, the received merged message is stored in the cloud
// and needs to be downloaded first.
if (message.type === TencentCloudChat.TYPES.MSG_MERGER && message.payload.downloadK
  let promise = chat.downloadMergerMessage(message);
  promise.then(function(imResponse) {
    // After the download is successful
    // the SDK will update information such as `message.payload.messageList`.
    console.log(imResponse.data);
  }).catch(function(imError) {
    // Download failed
    console.warn('downloadMergerMessage error:', imError);
  });
}
```

Forwarding Messages One by One

To forward a single message, create a message identical to the original message through the `createForwardMessage` API first, and then call the `sendMessage` API to send the message.

API

```
chat.createForwardMessage(options);
```

Parameters

The `options` parameter is of the `Object` type. It contains the following attribute values:

Name	Type	Description
to	String	`userID` or `groupID` of the message receiver
conversationType	String	Conversation type. Valid values: `TencentCloudChat.TYPES.CONV_C2C` (one-to-one conversation) `TencentCloudChat.TYPES.CONV_GROUP` (group conversation)
priority	String	Message priority. If messages in a group exceed the frequency limit, the backend will deliver high-priority messages first. Supported enumerated values: `TencentCloudChat.TYPES.MSG_PRIORITY_HIGH`

©2013-2025 Tencent Cloud International Pte. Ltd.

		`TencentCloudChat.TYPES.MSG_PRIORITY_NORMAL` (default)
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOW`
		`TencentCloudChat.TYPES.MSG_PRIORITY_LOWEST`

payload	Message	Message instance
cloudCustomData	String	Custom message data, which is saved in the cloud, will be sent to the receiver, and can still be pulled after the application is uninstalled and reinstalled.

Return value

`Message`

Examples

```
let forwardMessage = chat.createForwardMessage({
  to: 'user1',
  conversationType: TencentCloudChat.TYPES.CONV_C2C,
  payload: message, // Message instance for the received or sent message
  // cloudCustomData: 'your cloud custom data'
});
// 2. Send the message.
let promise = chat.sendMessage(forwardMessage);
promise.then(function(imResponse) {
  // Message sent successfully
  console.log(imResponse);
}).catch(function(imError) {
  // Failed to send the message
  console.warn('sendMessage error:', imError);
});
```

©2013-2025 Tencent Cloud International Pte. Ltd.

Modifying Message Android&iOS&Windows&Mac

Last updated：2025-01-06 17:32:17

Feature Description

This feature enables any participant in a conversation to modify a successfully sent message in the conversation. The message will be synced to all the participants in the conversation once modified successfully.

Note：

This feature is supported only by the Enhanced edition on v6.2 or later.

Effect

You can use this API to change the `cloudCustomData` of a message, implementing features such as message reply and message quote as shown below:

API Description

Modifying a Message

A conversation participant can call `modifyMessage` (Java / Swift / Objective-C / C++) to modify a sent message in the conversation.

The SDK allows any conversation participant to modify a message in the conversation. You can add more restrictions at the business layer, for example, only allowing the message sender to modify the message.

Currently, the following information of a message can be modified:

`cloudCustomData` (Java / Swift / Objective-C / C++)

`V2TIMTextElem` (Java / Swift / Objective-C / C++)

`V2TIMCustomElem` (Java / Swift / Objective-C / C++)

`V2TIMLocationElem` (Java / Swift / Objective-C/ C++)

`V2TIMFaceElem` (Java / Swift / Objective-C / C++)

Sample code:

Java

Swift

Objective-C

©2013-2025 Tencent Cloud International Pte. Ltd.

C++

```
// The original message object in the conversation is `originMessage`.
// Modify the `cloudCustomData` information of the message object
originMessage.setCloudCustomData("modify_cloud_custom_data".getBytes());
// If the message is a text message, modify the text message content
if (V2TIMMessage.V2TIM_ELEM_TYPE_TEXT == originMessage.getElemType()) {
  originMessage.getTextElem().setText("modify_text");
}
V2TIMManager.getMessageManager().modifyMessage(originMessage, new V2TIMCompleteCall
  @Override
  public void onComplete(int code, String desc, V2TIMMessage message) {
    // After the message is modified, `message` is the modified message object.
  }
});
```

```
// The original message object in the conversation is `originMessage`.
var originMessage: V2TIMMessage!
// Modify the `cloudCustomData` information of the message object
originMessage.cloudCustomData = "modify_cloud_custom_data".data(using: .utf8)
// If the message is a text message, modify the text message content
if originMessage.elemType == .V2TIM_ELEM_TYPE_TEXT {
    originMessage.textElem?.text = "modify_text"
}
// modifyMessage
V2TIMManager.shared.modifyMessage(msg: originMessage) { code, desc, msg in
    // After the message is modified, `msg` is the modified message object.
}
```

```
// Original message object in the conversation
V2TIMMessage *originMessage;
// Modify the `cloudCustomData` information of the message object
originMessage.cloudCustomData = [@"modify_cloud_custom_data" dataUsingEncoding:NSUT
// If the message is a text message, modify the text message content
if (V2TIM_ELEM_TYPE_TEXT == originMessage.elemType) {
    originMessage.textElem.text = @"modify_text";
}
[[V2TIMManager sharedInstance] modifyMessage:originMessage completion:^(int code, N
    // After the message is modified, `msg` is the modified message object.
}];
```

```
template <class T>
class CompleteCallback final : public V2TIMCompleteCallback<T> {
public:
    using InternalCompleteCallback =
```

```
        std::function<void(int, const V2TIMString&, const T&)>;

    CompleteCallback() = default;
    ~CompleteCallback() override = default;

    void SetCallback(InternalCompleteCallback complete_callback) { complete_callbac

    void OnComplete(int error_code, const V2TIMString& error_message, const T& valu
        if (complete_callback_) {
            complete_callback_(error_code, error_message, value);
        }
    }

private:
    InternalCompleteCallback complete_callback_;
};

// V2TIMMessage originMessage;
std::string str = u8"modify_cloud_custom_data";
// Modify the `cloudCustomData` information of the message object
originMessage.cloudCustomData = {reinterpret_cast<const uint8_t*>(str.data()), str.
if (originMessage.elemList.Size() == 1) {
    V2TIMElem* elem = originMessage.elemList[0];
    if (elem->elemType == V2TIMElemType::V2TIM_ELEM_TYPE_TEXT) {
        // If the message is a text message, modify the text message content
        auto textElem = static_cast<V2TIMTextElem*>(elem);
        textElem->text = "modify_text";
    }
}

auto callback = new CompleteCallback<V2TIMMessage>{};
callback->SetCallback([=](int error_code, const V2TIMString& error_message, const V
    // After the message is modified, `message` is the modified message object.
    delete callback;
});

V2TIMManager::GetInstance()->GetMessageManager()->ModifyMessage(originMessage, call
```

Message Modified Notification

Conversation participants call `addAdvancedMsgListener` (Java / Swift / Objective-C / C++) to add the advanced message listener.

After a message in the conversation is modified, all the participants will receive the `onRecvMessageModified` callback (Java / Swift / Objective-C / C++), which contains the modified message object.

Sample code:

Java

Swift

Objective-C

C++

```
V2TIMAdvancedMsgListener advancedMsgListener = new V2TIMAdvancedMsgListener() {
  // Notification of the message content modification
  @Override
  public void onRecvMessageModified(V2TIMMessage msg) {
      // `msg` is the modified message object.
  }
};
// Add a message listener
V2TIMManager.getMessageManager().addAdvancedMsgListener(advancedMsgListener);
```

```
// Add a message listener
V2TIMManager.sharedInstance().addAdvancedMsgListener(self)
/// Notification of the message content modification
func onRecvMessageModified(_ msg: V2TIMMessage) {
    // `msg` is the modified message object.
 }
```

```
// Add a message listener
[[V2TIMManager sharedInstance] addAdvancedMsgListener:self];
/// Notification of the message content modification
- (void)onRecvMessageModified:(V2TIMMessage *)msg {
    // `msg` is the modified message object.
}
```

```
class AdvancedMsgListener final : public V2TIMAdvancedMsgListener {
public:
    // Notification of the message content modification
    void OnRecvNewMessage(const V2TIMMessage& message) override {
        // `message` is the modified message object.
    }
    // Other members ...
};

// Note that `advancedMsgListener` should not be released before the IM SDK is unin
// otherwise the message callback cannot be called.
AdvancedMsgListener advancedMsgListener;
V2TIMManager::GetInstance()->GetMessageManager()->AddAdvancedMsgListener(&advancedM
```

©2013-2025 Tencent Cloud International Pte. Ltd.

JavaScript

Last updated：2025-01-07 15:20:40

Feature Overview

This feature enables any member in a conversation to modify a successfully sent message in the conversation. The message will be synced to all the members in the conversation once modified successfully.

UI Display

You can use the message modification API to change the `cloudCustomData` of a message, enabling features such as message replies and references as shown below:

©2013-2025 Tencent Cloud International Pte. Ltd.

Modifying a Message

This API is used to modify a message. After a user modifies a message successfully, both the user and the receiver (one-to-one) or group members (Group) will receive the `MESSAGE_MODIFIED` event.

Note:

1. It does not support modifying online messages or AVChatRoom messages. Do not modify the `random/sequence/time` fields of a message.
2. It only supports modifying text, custom, geographical location, and emoji messages.
3. If a message is modified by another member during modification, the SDK will return the error code 2480, indicating that a conflict occurred while modifying the message.
4. It supports for modifying the `cloudCustomData` field of all types of messages.

©2013-2025 Tencent Cloud International Pte. Ltd.

API

```
chat.modifyMessage(message);
```

Parameters

Name	Type	Description
message	Message	Message instance

Return value

`Promise`

Examples

```
// Listen for the `MESSAGE_MODIFIED` event
// which will be delivered by the SDK after a message is modified successfully
let onMessageModified = function(event) {
  // event.data - An array that stores modified Message objects - [Message]
};
chat.on(TencentCloudChat.EVENT.MESSAGE_MODIFIED, onMessageModified);

// Change the text content of `txtMessage` to `Hello Tencent`
txtMessage.payload.text = "Hello Tencent";
```

©2013-2025 Tencent Cloud International Pte. Ltd.