Chat

Demo Zone

Product Documentation

Contents

Demo Zone

Free Demos

Last updated：2024-07-09 15:25:38

You can click the button to try the Web Demo：

Quick Run

Overview

Last updated：2024-05-13 16:35:41

Please choose your development platform to quickly run the Chat Demo:

Android

iOS

Web(React)

Electron

uni-app

Web & H5 (Vue)

Unity

UE

React Native

Flutter

Quickly Integrating into Your Existing Native App with Flutter

Android

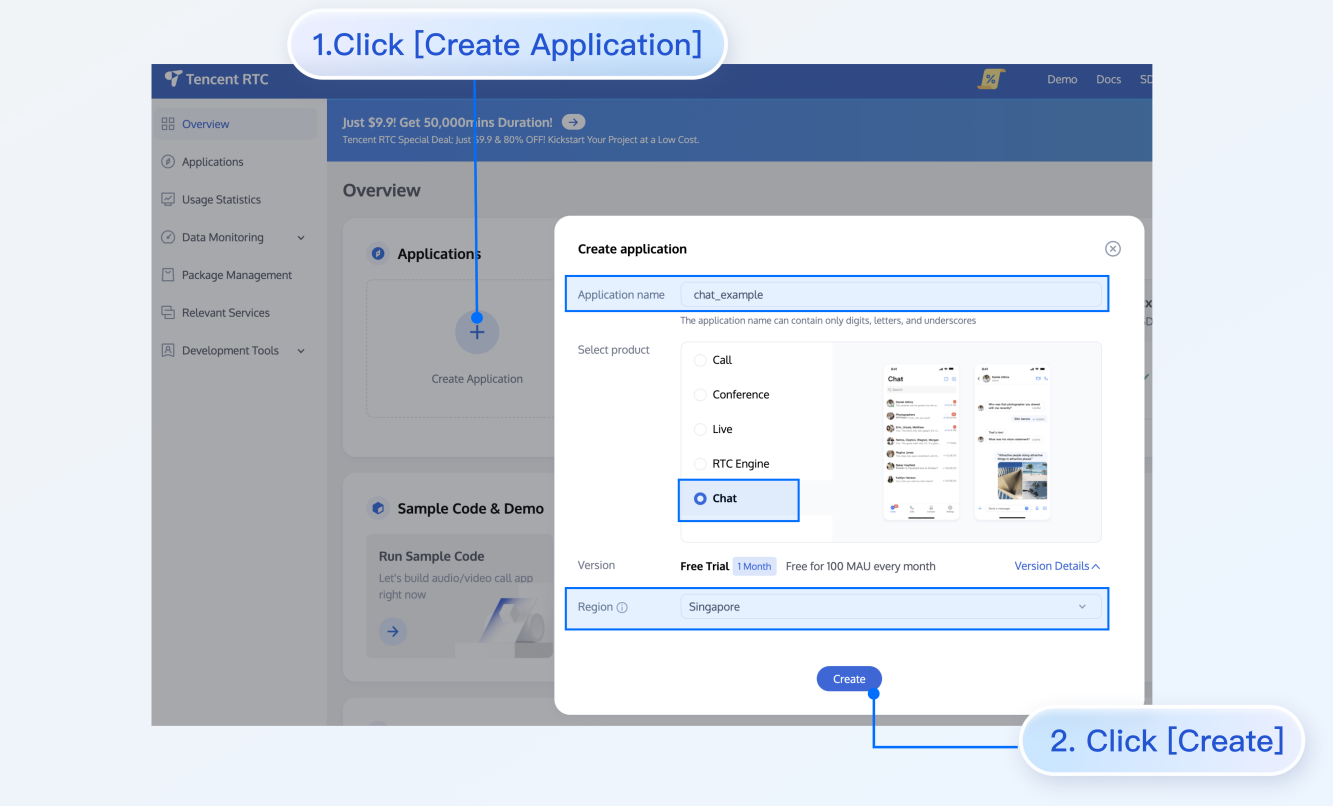
Last updated：2024-11-04 16:57:57

This document mainly describes how to quickly run the Chat Demo.

Directions

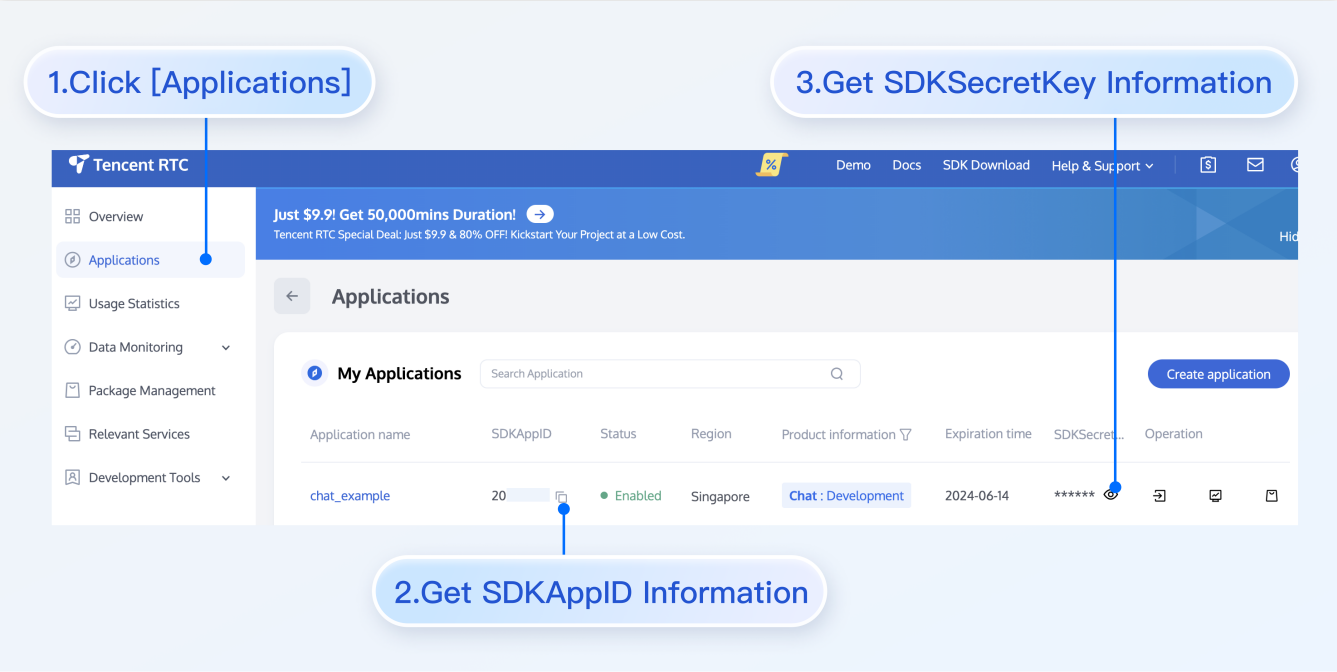
1. Create an App

1. Log in to the Tencent RTC Console. If you already have an app, record its SDKAppID and SDKSecretKey and jump directly to the next section.
2. Click `Create Application` , enter your Application name, product, Region, and click `Create` .



2. Obtain SDKAppID and SDKSecretKey

After creation, you can view the newly created app's Status, SDKAppID, Expiration time, etc., on the Applications page:



Record the SDKAppID and SDKSecretKey from the Application Information.

Danger:

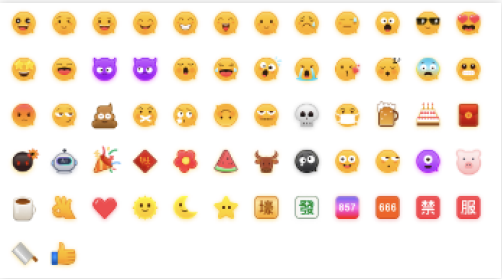
Keep the SDKSecretKey properly to prevent disclosure.

3. Download and Configure the Demo

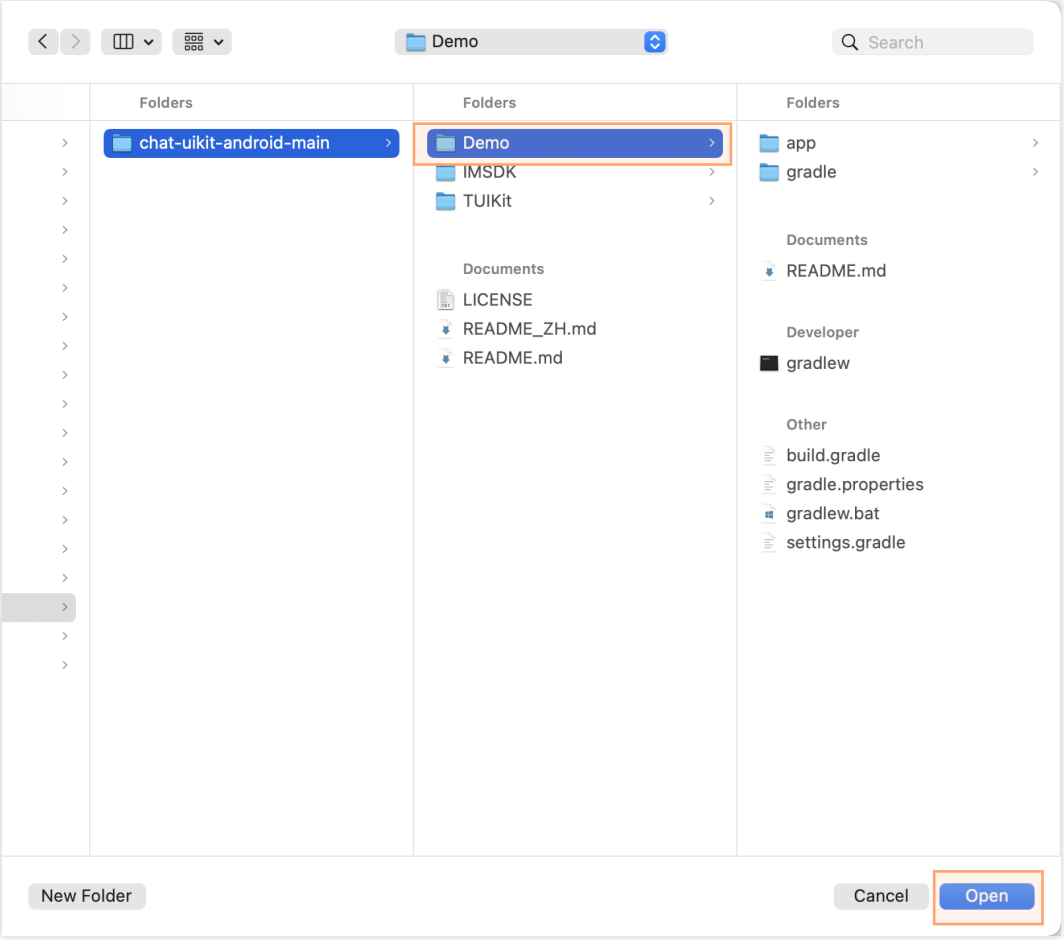
1. Download the Android demo project from Github.

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.



2. Open the Chat project through Android Studio:

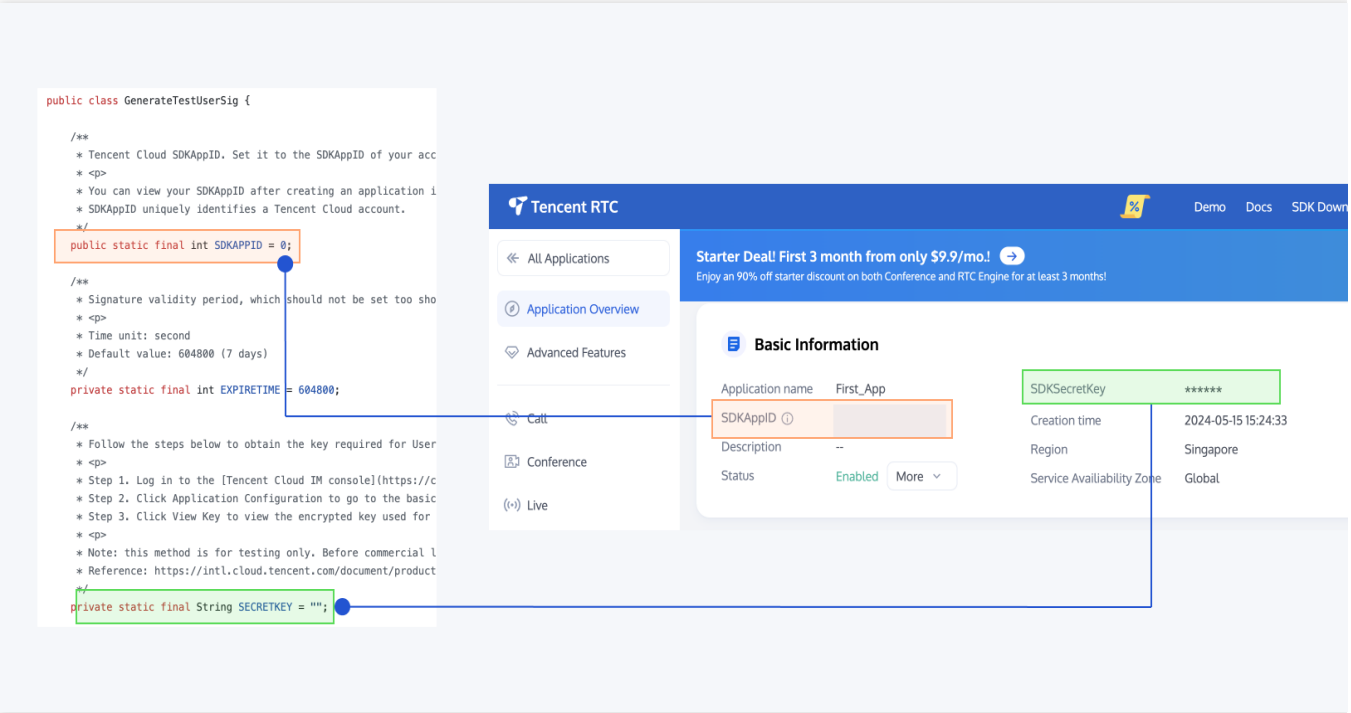


3. Find the `GenerateTestUserSig.java` file. The path is： `Android/Demo/app/src/main/java/com/tencent/qcloud/tim/demo/signature/GenerateTestUserSig.java`

4. Set the relevant parameters:

SDKAPPID: set it to the SDKAppID obtained above.

SECRETKEY: set it to the SDKSecretKey obtained above.



Warning：

In this document, the method to obtain `UserSig` is to configure a `SECRETKEY` in the client code. In this method, the `SECRETKEY` is vulnerable to decompilation and reverse engineering. Once your `SECRETKEY` is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an app-oriented API. When `UserSig` is needed, your app can send a request to the business server to obtain a dynamic `UserSig`. For more information, see How to Generate UserSig on the Server.

To respect the copyright of emoji design, the downloaded demo project does not contain sliced images of major emoji elements. You can use your local emoji packs to configure code. Unauthorized use of the emoji pack in the Chat demo may infringe on the design copyright.

4. Compile and Run the Demo

Import the demo project with Android Studio, and then compile and run it. For more information, see the file `README.md` in the corresponding directory of the demo project cloned above.

Environment requirements

Android Studio-Chipmunk

Gradle-6.7.1

Android Gradle Plugin Version-4.2.0

kotlin-gradle-plugin-1.5.31

Note:

The demo is integrated with the audio/video call feature by default. However, the TRTC SDK on which the audio/video call feature relies currently does not support simulators. Please use real devices for demo running or debugging.

Experience basic features

Create User Account

If you have successfully run the Demo by following the above steps, you can start experiencing the basic features. First, you need to create a user account. There are many ways to do this, for example, by registering through log in to Demo on the client side or creating some in the console. You can choose any method that suits you.

Client Registration

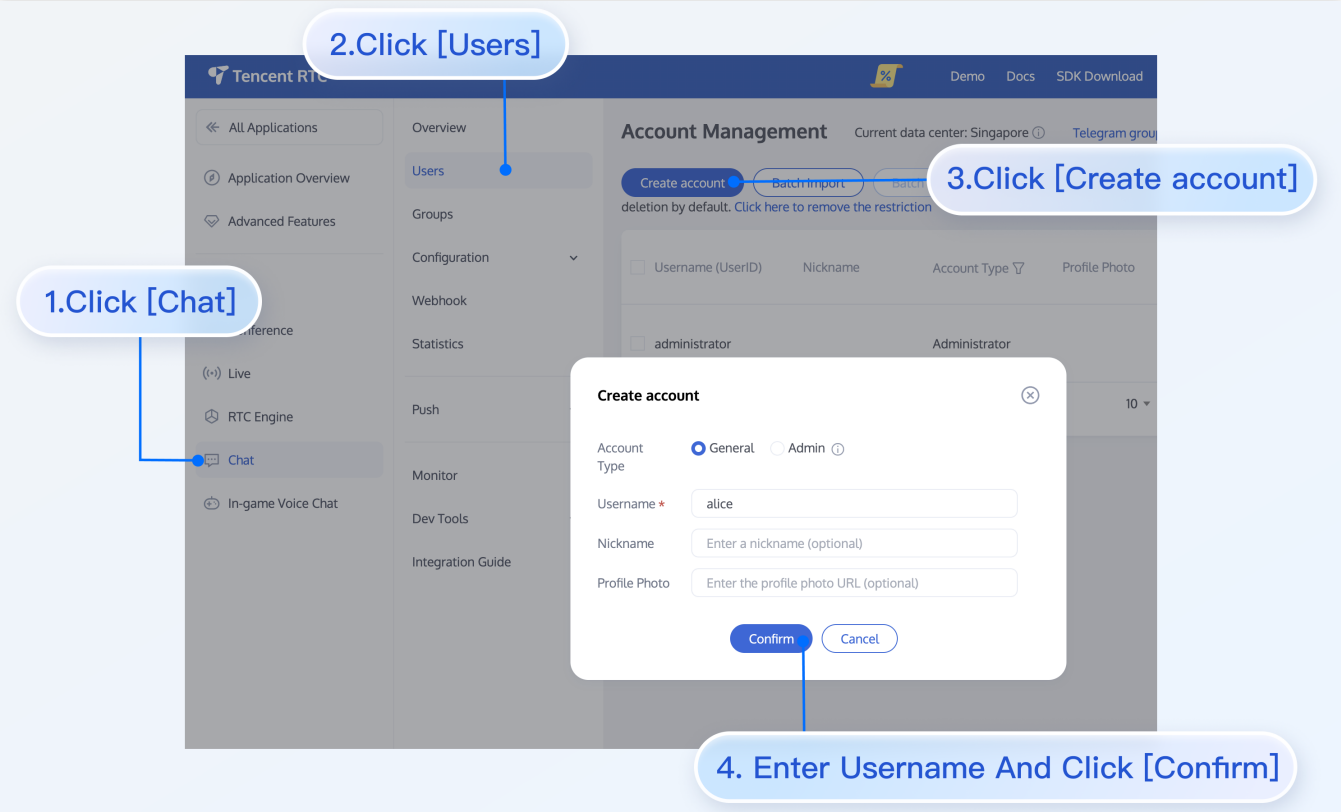
Simply log in to the Demo with several different accounts.

Create in Console

The steps are as follows:

1. Click to enter the application you created above, and you will see the Chat product entry on the left sidebar, click to enter.
2. After entering the Chat Product subpage, click on `Users` to go to the User Management Page.
3. Click `Create account`, a pop-up will appear for you to fill in the account creation information. If you are just a regular member, we recommend you choose the `General` type. Although `Nickname` is not mandatory, we still suggest you set it. If it's inconvenient to display `userID` on the interface, you can identify different users through `Nickname`.

The details are as follows:



Note:

Sending messages involves at least two users, so at this step, you need to create at least 2 accounts. Please note down the userID of these 2 accounts for subsequent steps.

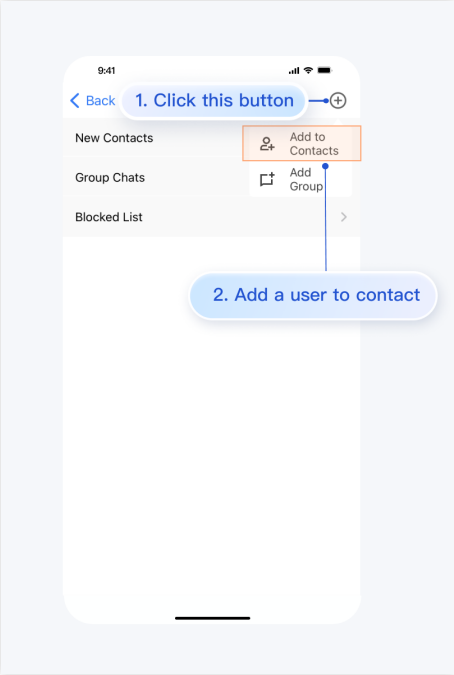
Add to Contacts

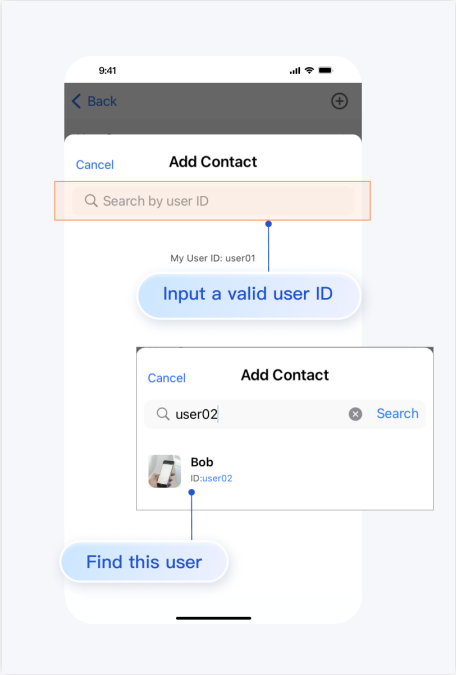
After switching to the Contacts interface:

1. Click the `+` button in the top right corner of the interface, and in the submenu, select `Add to Contacts`.
2. Enter a valid userID and search for the user. If you have already created an account in the console, you can go to the console `Account Management` page to get a valid userID. Page path: Applications > Your App > Chat > Users > Account Management.
3. Add user as contact.

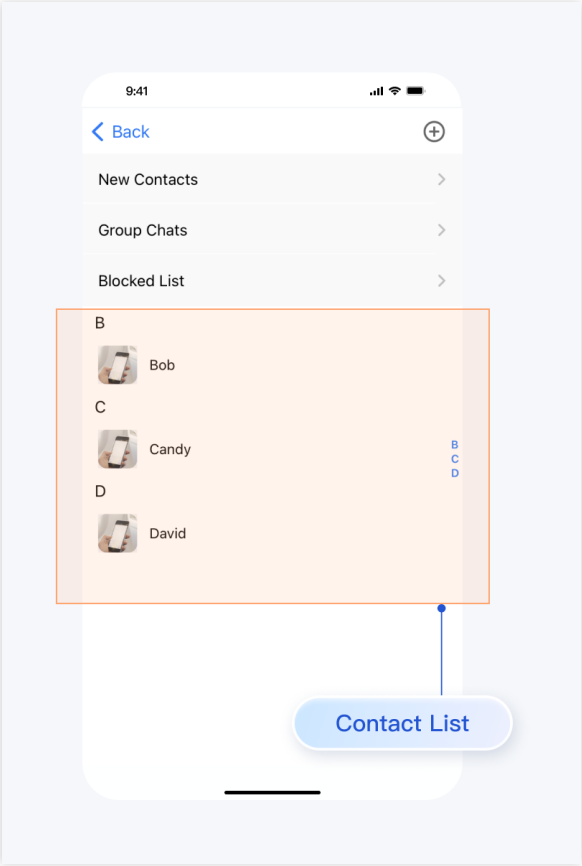
The steps are shown below:

Click [Add to Contacts]	Search user	Send



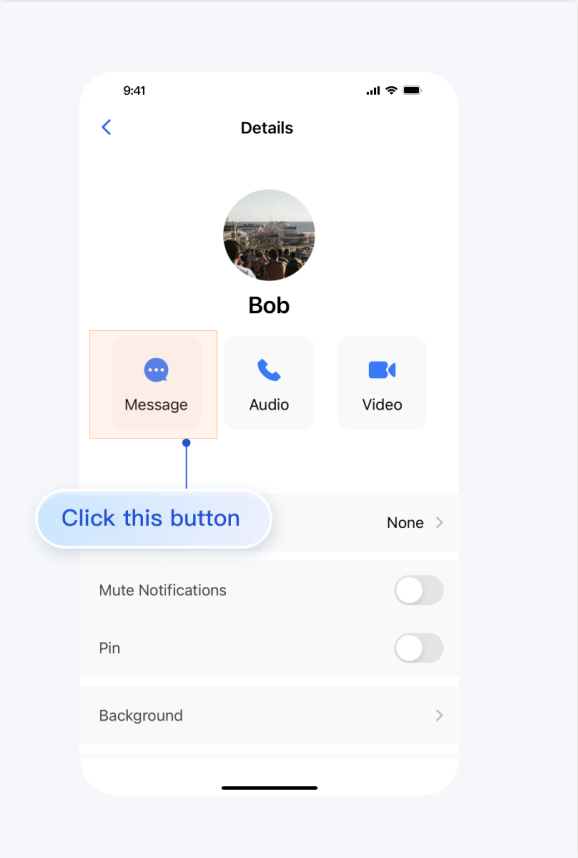


After successfully adding, the user will appear in the contact list:

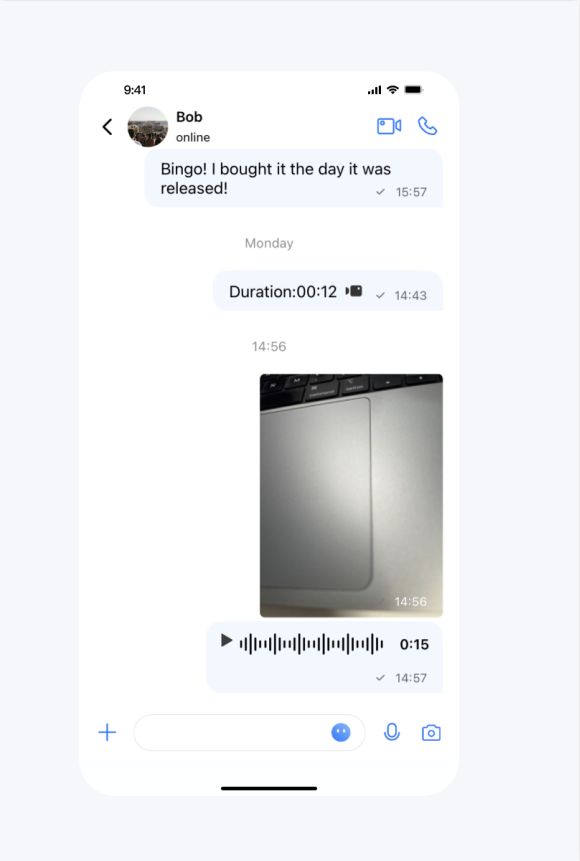


Sending Messages

Select a user, click `Message` to enter the message interface:



Next, you can send messages, voice, images, and make audio/video calls with the user in the message interface:


9:41
Bob
online
Bingo! I bought it the day it was released!
15:57
Monday
Duration:00:12
14:43
14:56
14:56
0:15
14:57

iOS

Last updated：2025-05-29 10:29:23

This document mainly describes how to quickly run the Chat Demo.

Directions

1. Create an App

1. Log in to the Tencent RTC Console. If you already have an app, record its SDKAppID and SDKSecretKey and jump directly to the next section.
2. Click `Create Application` , enter your Application name, product, Region, and click `Create` .

2. Obtain SDKAppID and SDKSecretKey

After creation, you can view the newly created app's Status, SDKAppID, Expiration time, etc., on the Applications page:

Record the SDKAppID and SDKSecretKey from the Application Information.

Danger:

Keep the SDKSecretKey properly to prevent disclosure.

3. Download and Configure the Demo

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.

Download Swift Demo

1. Download Swift Chat UIKit Project.
2. Open the project in the corresponding terminal directory and find the `GenerateTestUserSig.swift` file.

iOS path: Chat_UIKit/Swift/TUIKitDemo/TUIKitDemo/Private/GenerateTestUserSig.swift

3. Set the relevant parameters in the `GenerateTestUserSig.swift` file:

Set public_SDKAPPID to the actual Application SDKAppID obtained in Step 1.

Set public_SECRETKEY to the actual key information obtained in Step 2.

Download Objective-C Demo

1. Download the iOS demo project from Github.
2. Open the project in the terminal directory and find the `GenerateTestUserSig.h` file. The path is: `chat-uikit-ios-main/Demo/TUIKitDemo/Private/GenerateTestUserSig.h`
3. Set the relevant parameters:

SDKAPPID: set it to the SDKAppID obtained above.

SECRETKEY: set it to the SDKSecretKey obtained above.

Warning：

1. In this document, the method to obtain `UserSig` is to configure a `SECRETKEY` in the client code. In this method, the `SECRETKEY` is vulnerable to decompilation and reverse engineering. Once your `SECRETKEY` is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.
2. The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an app-oriented API. When `UserSig` is needed, your app can send a request to the business server to obtain a dynamic `UserSig`. For more information, see How to Generate UserSig on the Server.
3. To respect the copyright of emoji design, the downloaded demo project does not contain sliced images of major emoji elements. You can use your local emoji packs to configure code. Unauthorized use of the emoji pack in the Chat demo may infringe on the design copyright.

4. Compile and Run the Demo

See the file `README.md` in the corresponding directory of the demo project cloned above.

1. Run the following command on the terminal to check the pod version:

```
pod --version
```

If the system indicates that no pod exists or that the pod version is earlier than 1.7.5, run the following commands to install the latest pod.

```
//Change Gem Source
gem sources --remove https://rubygems.org/
gem sources --add https://gems.ruby-china.com/
//Install pod
sudo gem install cocoapods -n /usr/local/bin
```

```
// If multiple versions of Xcode are installed, run the following command to
choose an Xcode version (usually the latest one):
sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer
//Update Pod Local Repository
pod setup
```

2. Run the following commands on the terminal, and open the directory where Podfile is located to install dependent libraries:

Swift

Objective-C

```
cd Swift/TUIKitDemo
pod install
```

```
cd chat-uikit-ios-main/Demo
pod install
```

If installation fails, run the following command to update the local CocoaPods repository list:

```
pod repo update
```

3. Compile and run the demo:

Swift project, enter the `Swift/TUIKitDemo` folder, open the `TUIKitDemo.xcworkspace` and compile and run.

Objective-C project, enter the `chat-uikit-ios-main/Demo` folder, open the `TUIKitDemo.xcworkspace` and compile and run.

Note：

The demo is integrated with the audio/video call feature by default. However, the TRTC SDK on which the audio/video call feature relies currently does not support simulators. Please use real devices for demo running or debugging.

Experience basic features

Create User Account

If you have successfully run the Demo by following the above steps, you can start experiencing the basic features. First, you need to create a user account. There are many ways to do this, for example, by registering through log in to Demo on the client side or creating some in the console. You can choose any method that suits you.

Client Registration

Simply log in to the Demo with several different accounts.

Create in Console

The steps are as follows:

1. Click to enter the application you created above, and you will see the Chat product entry on the left sidebar, click to enter.
2. After entering the Chat Product subpage, click on `Users` to go to the User Management Page.
3. Click `Create account`, a pop-up will appear for you to fill in the account creation information. If you are just a regular member, we recommend you choose the `General` type. Although `Nickname` is not mandatory, we still suggest you set it. If it's inconvenient to display `userID` on the interface, you can identify different users through `Nickname`.

The details are as follows:

Note:

Sending messages involves at least two users, so at this step, you need to create at least 2 accounts. Please note down the userID of these 2 accounts for subsequent steps.

Add to Contacts

After switching to the Contacts interface:

1. Click the `+` button in the top right corner of the interface, and in the submenu, select `Add to Contacts`.
2. Enter a valid userID and search for the user. If you have already created an account in the console, you can go to the console `Account Management` page to get a valid userID. Page path: Applications > Your App > Chat > Users > Account Management.
3. Add user as contact.

The steps are shown below:

Click [Add to Contacts]	Search user	Send contact request

After successfully adding, the user will appear in the contact list:

Sending Messages

Select a user, click `Message` to enter the message interface:

Next, you can send messages, voice, images, and make audio/video calls with the user in the message interface:

React

Last updated：2025-03-21 10:59:15

This article will introduce how to quickly implement a chat demo. You will complete the following key steps within 10 minutes and ultimately obtain a chat feature with a full user interface. Before you start, you can directly experience the chat below or **Try On CodeSandbox**.

Environment Requirements

Node.js version 16+
npm (use a version that matches the Node version in use)

Running Demo

Step 1: Download the demo source code

MacOS
Windows

```
// Run the code in CLI
git clone https://github.com/TencentCloud/chat-uikit-react
// Go to the project
cd chat-uikit-react/examples/sample-chat
// Install dependencies of the demo
npm install
```

```
// Run the code in CLI
git clone https://github.com/TencentCloud/chat-uikit-react
```

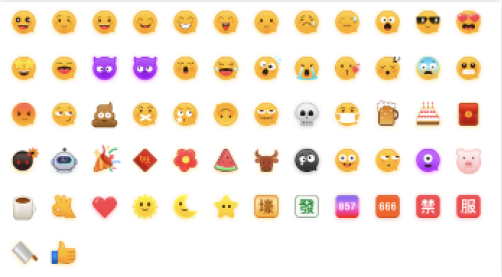
```
// Go to the project
cd chat-uikit-react/examples/sample-chat
```

```
// Install dependencies of the demo
npm install
```

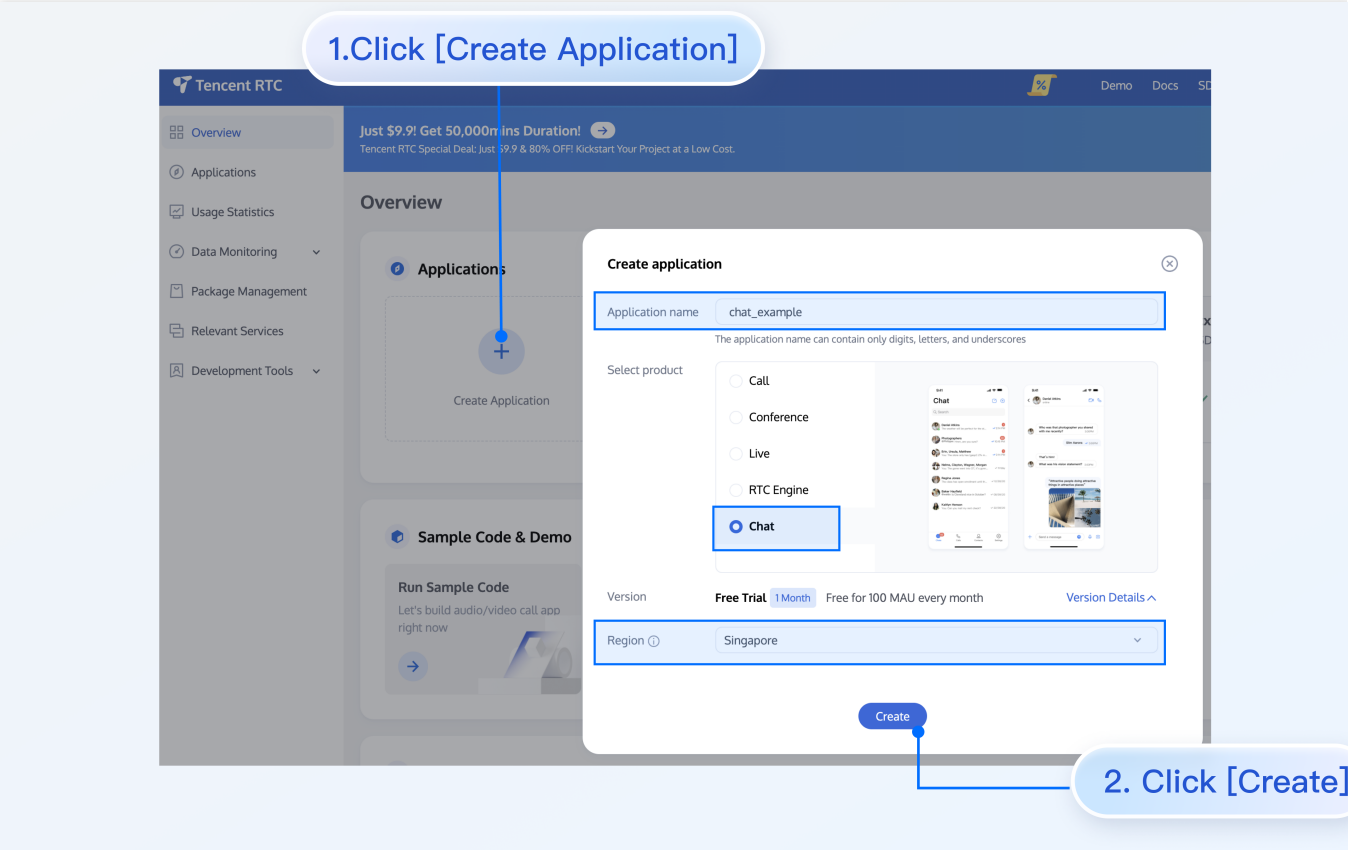
Step 2: Configure the demo

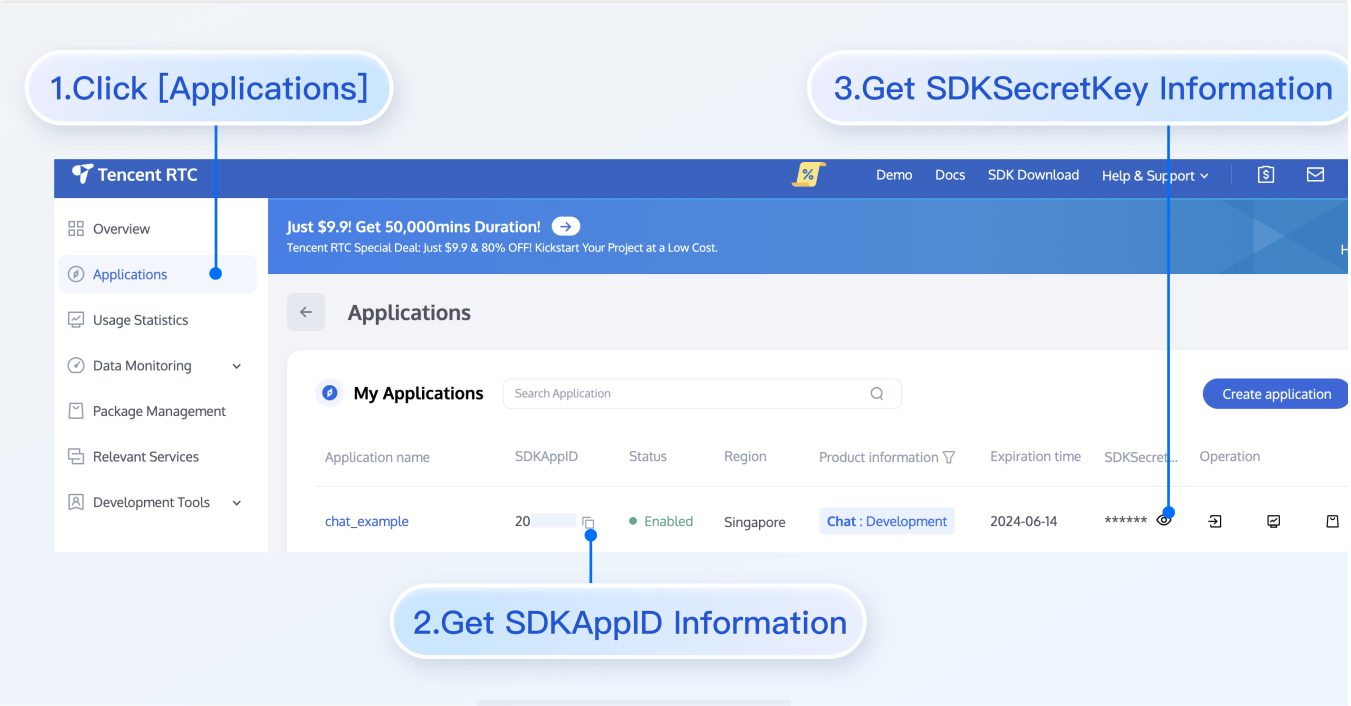
Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.

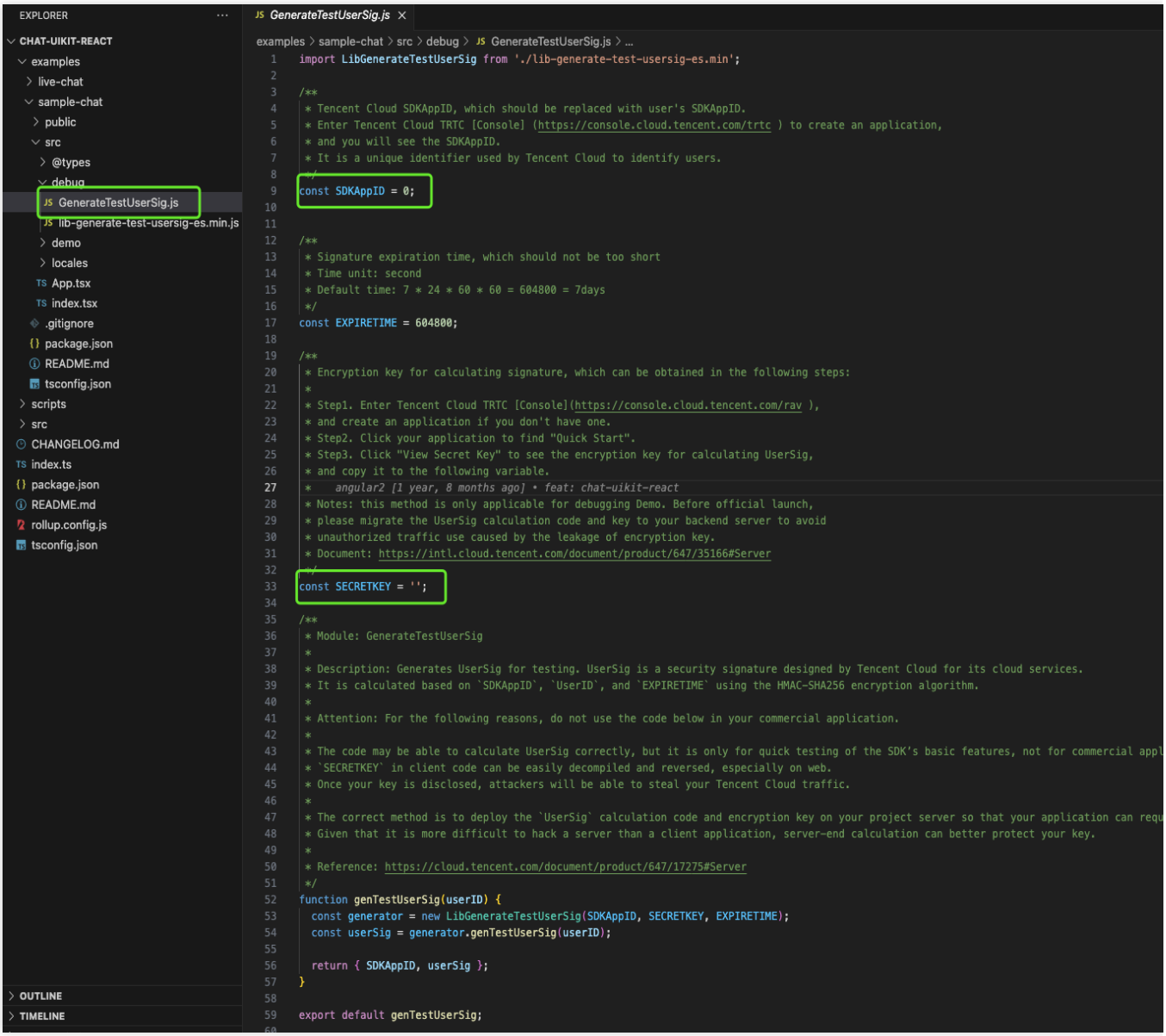


1. Open the `examples/sample-chat` project and find the `GenerateTestUserSig.js` file in the path `./examples/sample-chat/src/debug/GenerateTestUserSig.js`.
2. In the `GenerateTestUserSig.js` file, set `SDKAPPID` and `SECRETKEY`. Their values can be obtained in the Chat console. Click the target application tab to enter its configuration page.





3. Copy the key information and save it to the `./examples/sample-chat/src/debug/GenerateTestUserSig.js` file.



Note:

This document mentions a method for generating UserSig by configuring a secret key in the client code. This method makes the secret key susceptible to decompilation and reverse engineering. Once your key is compromised, attackers can misappropriate your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo and debugging features.**

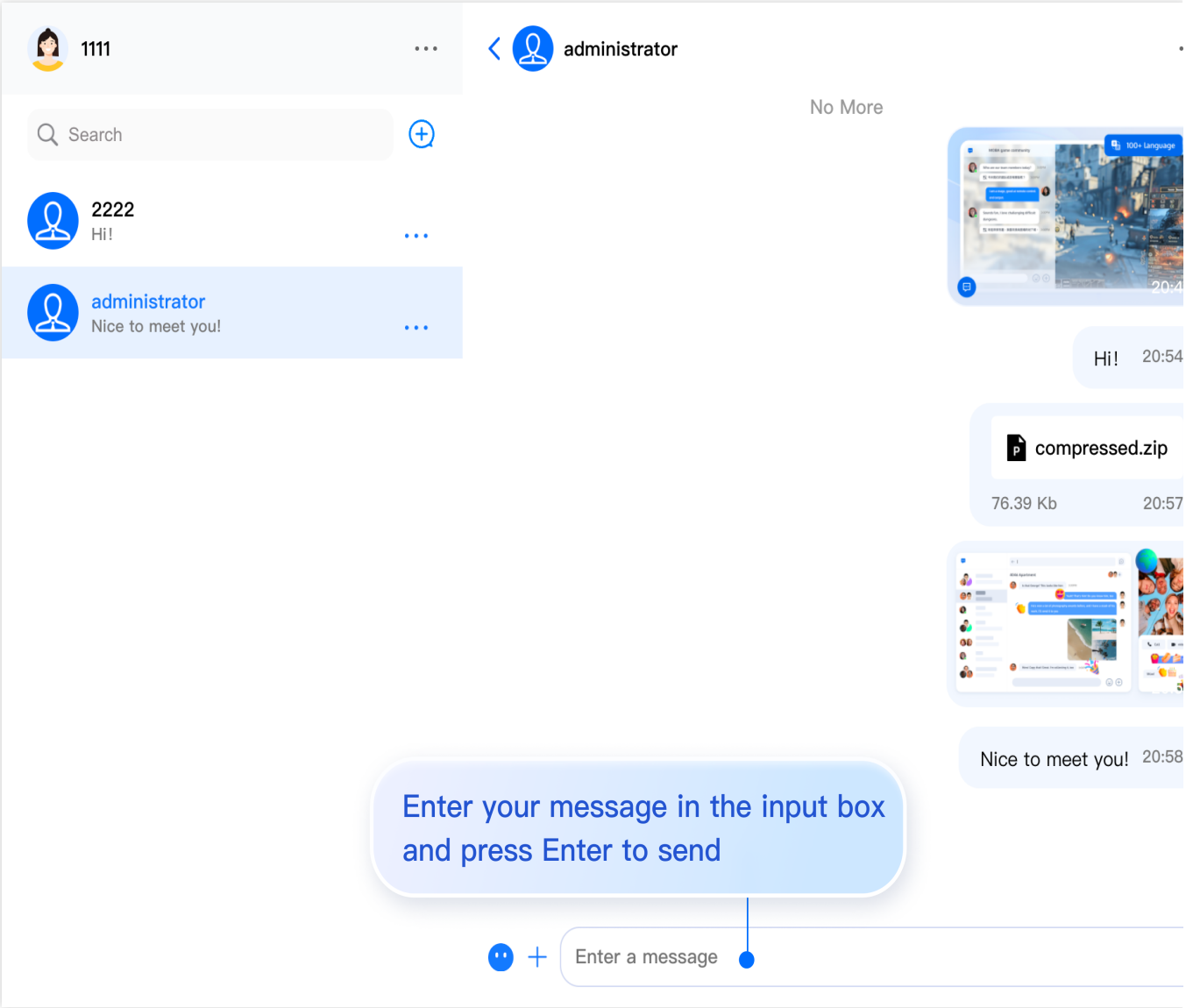
The correct way to generate UserSig is to integrate the UserSig computation code into your server and provide an API for the app. When UserSig is needed, your app should request a dynamic UserSig from the business server. For more information, see How to Generate a UserSig on the Server.

Step 3: Start the project

```
# Launch the project
npm run start
```

Step 4: Send your first message

Enter a message in the input box and press Enter to send.



Integrating chat-uikit-react

If you wish to integrate chat-uikit-react into your project, please go to Integration chat-uikit-react.

Contact Us

Join the Telegram technical discussion group or WhatsApp discussion group, enjoy the support of professional engineers, and solve your difficulties.

Electron

Last updated：2024-11-04 16:57:57

This document describes how to quickly run the Tencent Cloud Chat demo for Electron and integrate the Electron SDK.

Environment Requirements

Platform	Version
Electron	v13.1.5 or later
Node.js	v14.2.0

Supported Platforms

Currently, both macOS and Windows platforms are supported.

Trying Out Demo

Before integration, you can try out our demo to quickly understand the capabilities of the Tencent Cloud Chat SDK for Electron.

Prerequisites

You have signed up for a Tencent Cloud account and completed identity verification.

Directions

Step 1. Create an app

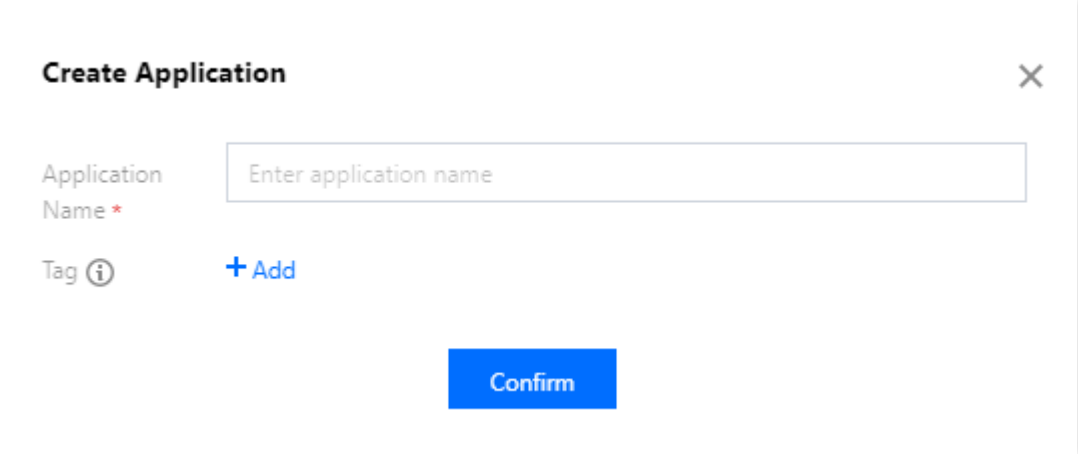
1. Log in to the Chat console.

Note:

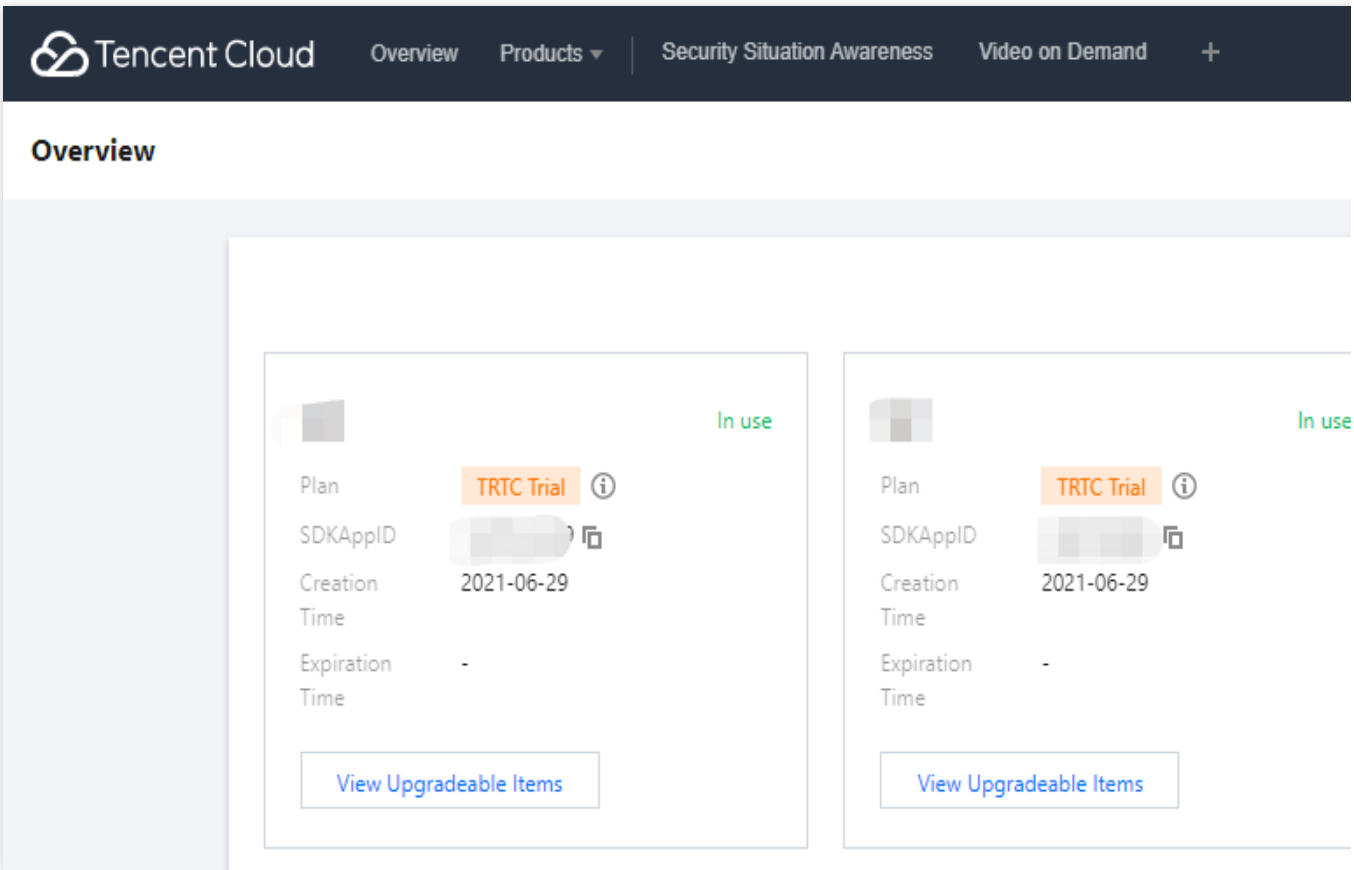
If you already have an app, record its SDKAppID and obtain key information.

A Tencent Cloud account can create a maximum of 300 Chat apps. If you want to create a new app, disable and delete an unwanted app first. **Once an app (along with its SDKAppID) is deleted, the service it provides and all its data are lost. Proceed with caution**.

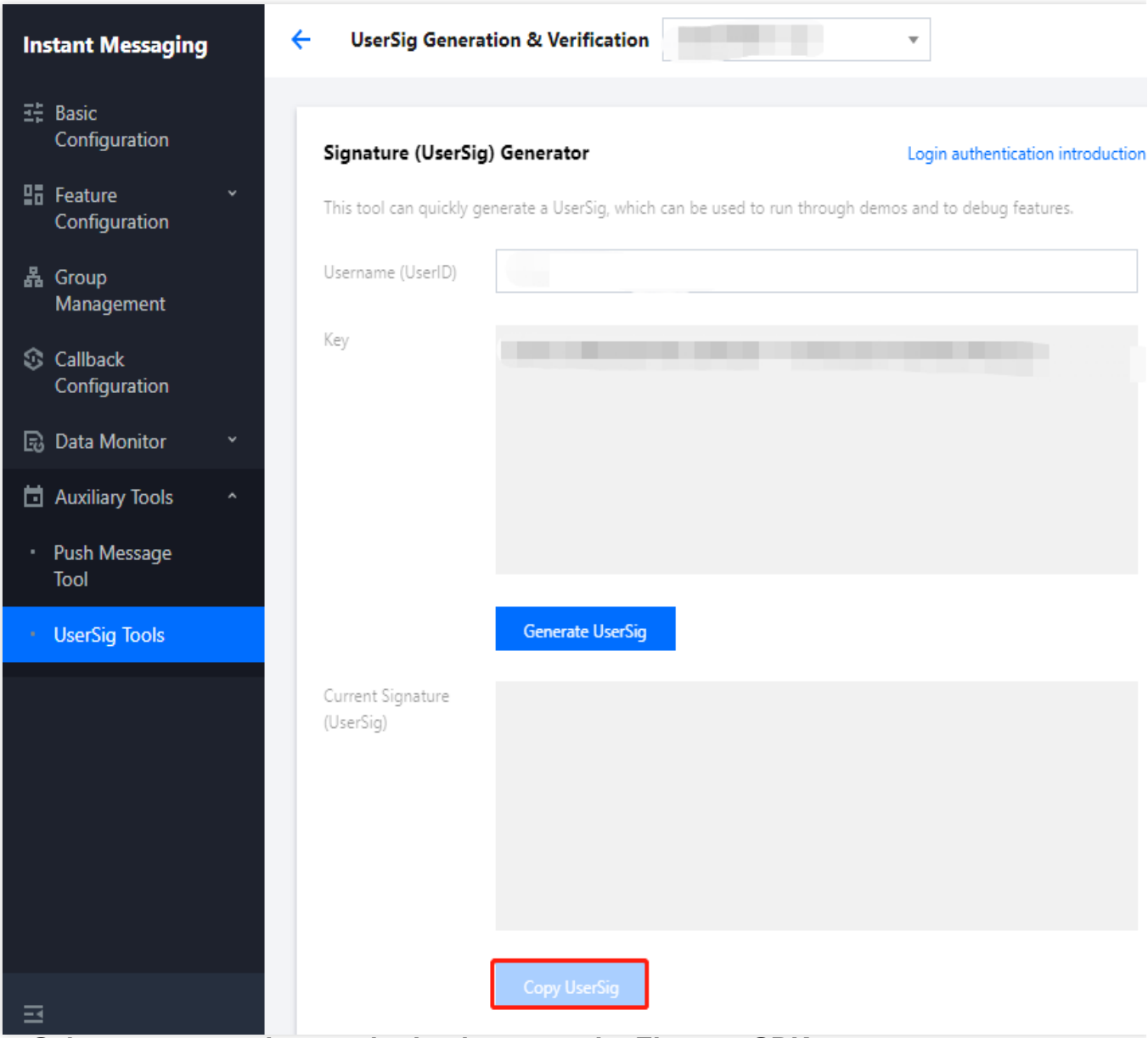
2. Click **Create Application**, enter your app name, and click **Confirm**.



3. After creation, you can see the status, service version, SDKAppID, tag, creation time, and expiry time of the new app on the overview page of the console. Record the SDKAppID.



4. Click the created app. In the left sidebar, click **Auxiliary Tools** > **UserSig Tools** to create a UserID and the corresponding UserSig. Then copy the UserSig for future login.



Step 2. Select an appropriate method to integrate the Electron SDK

Tencent Cloud Chat offers two integration schemes:

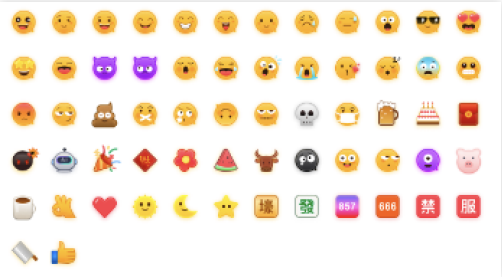
Integration Scheme	Applicable Scenario
Using a demo	The Chat demo includes all chat features and provides open-source code. If you need to implement chat scenarios, you can use the demo for secondary development. Try it out here.
Self implementation	Implement Chat on your own if the demo does not meet your UI requirements.

To help you better understand Chat SDK APIs, sample APIs are provided here.

Step 3. Use the demo

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.



1. Clone the source code of the Chat Electron demo to the local system.

```
git clone https://github.com/TencentCloud/tc-chat-demo-electron.git
```

2. Install project dependencies.

```
// Root directory of the project
npm install

// Rendering process directory
cd src/client
npm install
```

3. Run the project.

```
// Root directory of the project
npm start
```

4. Build the project.

```
// Build the project in macOS
npm run build:mac
// Build the project in Windows
npm run build:windows
```

Note:

In the demo, the main process directory `src/app/main.js` , and the rendering process directory is `src/client` . If any problem occurs during running, see the FAQs for troubleshooting first.

Step 4. Self implementation

Installing the Electron SDK

Install the latest version of the Electron SDK as follows.

Run the following command:

```
npm install im_electron_sdk
```

** Initializing the SDK**

1. Pass in your `sdkAppID` in `TimMain` .

```
// Main process
const TimMain = require('im_electron_sdk/dist/main')

const sdkappid = 0;// You can apply for it in the Chat console.
const tim = new TimMain({
  sdkappid:sdkappid
})
```

2. Call `TIMInit` to initialize the SDK.

```
// Rendering process
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
// Initialize the component
timRender.TIMInit()
```

3. Log in as a test user.

Log in with the test account initially generated in the console for login verification.

Call the `timRender.TIMLogin` method to log in as the test user.

If the returned `code` is `0` , the login is successful.

```
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
let {code} = await timRender.TIMLogin({
  userID:"userID",
  userSig:"userSig" // See how to generate a userSig
})
```

Note:

This account is for development and testing only. Before the application is launched, the correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see Generating UserSig.

Sending a message

The following sample shows how to send a text message. If the returned `code` is `0`, the message is sent successfully.

Sample code:

```
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
let param:MsgSendMessageParamsV2 = {    // param of TIMMsgSendMessage
    conv_id: "conv_id",
    conv_type: 1,
    params: {
        message_elem_array: [{
            elem_type: 1,
            text_elem_content:'Hello Tencent!',

        }],
        message_sender: "senderID",
    },
    callback: (data) => {}
  }
let {code} = await timRender.TIMMsgSendMessageV2(param);
```

Note:

If sending the message fails, it may be that your `sdkAppID` does not support sending messages to strangers. In this case, you can disable the friend relationship chain check in the console for testing.

Getting the conversation list

Log in with the other test account to pull the conversation list.

Common use cases include:

Get the conversation list upon application start and listen for the persistent connection to update the conversation list in real time.

```
let param:getConvList = {
            userData:userData,
        }
let data:commonResult<convInfo[]> = await timRenderInstance.TIMConvGetConvList(para
```

At this point, you can see the message sent by the other test account in the previous step.

Receiving a message

Common use cases include:

1. After a new conversation is opened on the UI, request a certain number of historical messages at a time to display the historical message list.
2. Listen for the persistent connection to receive messages in real time and add them to the historical message list.

Requesting the historical message list at a time

```
let param:MsgGetMsgListParams = {
        conv_id: conv_id,
        conv_type: conv_type,
        params: {
            msg_getmsglist_param_last_msg: msg,
            msg_getmsglist_param_count: 20,
            msg_getmsglist_param_is_remble: true,
        },
        user_data: user_data
    }
    let msgList:commonResult<Json_value_msg[]> = await timRenderInstance.TIMMsgGetM
```

Listening for new messages in real time

The following is the sample code for callback binding:

```
let param : TIMRecvNewMsgCallbackParams = {
            callback: (...args)=>{},
            user_data: user_data
        }
timRenderInstance.TIMAddRecvNewMsgCallback(param);
```

At this point, you have completed the Chat module development, and now users can send and receive messages and enter different conversations.

You can develop more features, such as group, user profile, relationship chain, offline push, and local search.

For detailed directions, see here.

FAQs

What platforms are supported?

Currently, both macOS and Windows platforms are supported.

How do I query error codes?

For Chat SDK API error codes, see Error Codes.

What should I do if the error `npm ERR! gyp ERR! stack TypeError: Cannot assign to read only property 'cflags' of object '#<Object>'` is reported during development environment installation?

Downgrade the node version to 16.18.1.

What should I do if the error `gypgyp ERR!ERR` is reported during development environment installation?

See gypgyp ERR!ERR! .

What should I do if the error `npm ERR! Fix the upstream dependency conflict, or retry` is reported when `npm install` is run?

In versions earlier than npm v7, dependency conflicts that occur during installation are automatically ignored.
In npmv7 or later versions, dependency conflicts will not be automatically ignored, and you need to manually enter a command to ignore them.
The command for ignoring dependency conflicts is as follows:

```
npm install --force
```

What should I do if the error `Error: error:0308010C:digital envelope routines::unsupported` is reported when `npm run start` is run?

Downgrade the node version to 16.18.1.

What should I do if the screen turns white when I run `npm run start` on a macOS client demo?

The error occurs because the rendering process code is not completely built and the port 3000 opened by the main process points to an empty page. The error will be resolved after the rendering process code is completely built and you refresh the window. Alternatively, you can run `cd src/client && npm run dev:react` and `npm run dev:electron` to start the rendering process and main process separately.

How do I use native modules in projects built with vue-cli-plugin-electron-builder?

For issues related to using native modules in projects built with vue-cli-plugin-electron-builder, see No native build was found for platform = xxx.

How do I use native modules in projects built with webpack?

For issues related to using native modules in projects built with webpack, see FAQs in the Windows environment.

What should I do if the error "Dynamic Linking Error" is reported?

Dynamic Linking Error. electron-builder configuration

```
    extraFiles:[
     {
       "from": "./node_modules/im_electron_sdk/lib/",
       "to": "./Resources",
       "filter": [
         "**/*"
       ]
     }
   ]
```

Getting `__dirname is not defined` when using electron-vite?

Since electron-vite does not support node integration and communicating between main and renderer processes in renderer process, Tencent Cloud Chat SDK needs to be written in `preload` for use. The code for Main process should be written in main process normally. For details, please refer to electron-vite documentation.

The usage is the same. Please refer to the example code of the document. Taking initialization as an example, the example code is as follows:

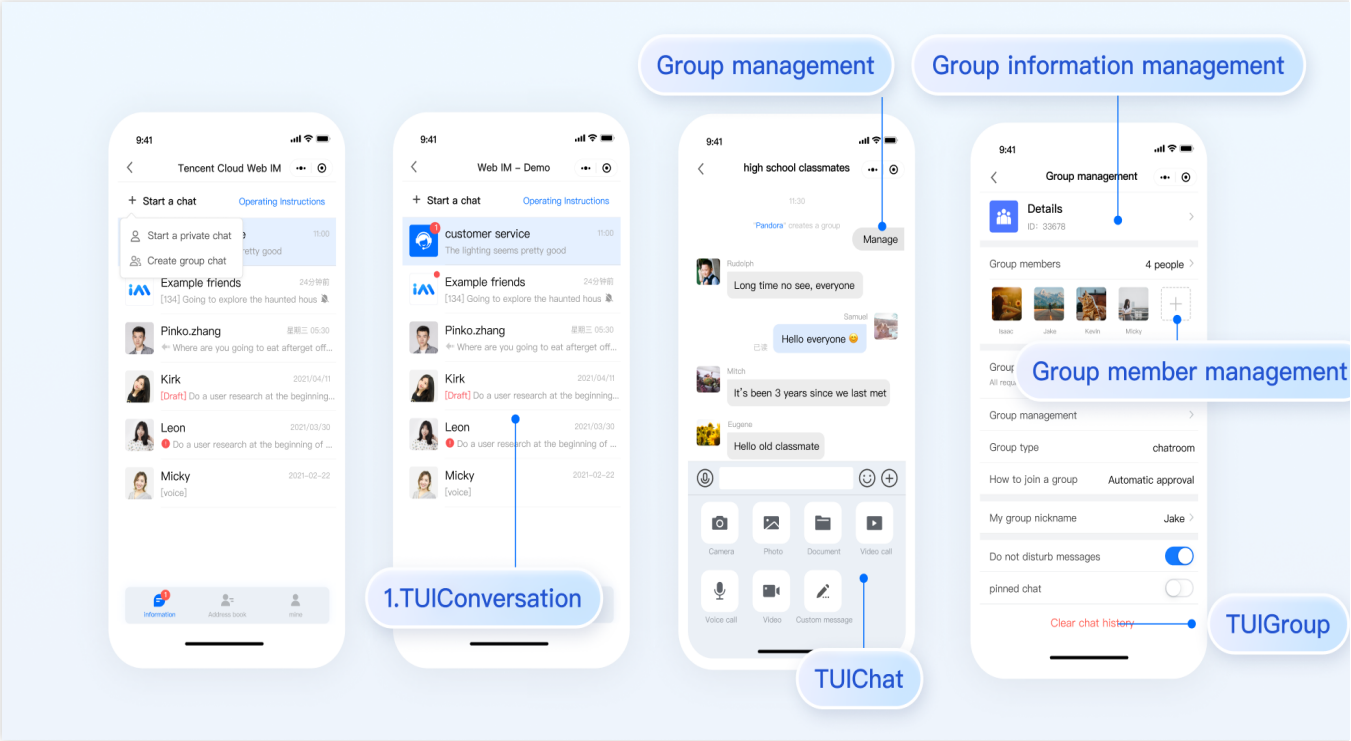
```
//The content of the main process is written to the main process
// main/index.ts (example path)
const TimMain = require('im_electron_sdk/dist/main')
const sdkappid = 0;
const tim = new TimMain({
   sdkappid:sdkappid
})
// Use chat sdk in preload
// preload/index.ts (example path)
import TimRender from 'im_electron_sdk/dist/renderer'
const timRender = new TimRender();
```

uniapp

Last updated：2024-11-04 16:57:57

Introduction to chat-uikit-uniapp

chat-uikit-uniapp (vue2 /vue3) is a uniapp UI component library based on Tencent Cloud Chat SDK. It provides universally used UI components that include Conversation, Chat, and Group components. Leveraging these meticulously crafted UI components, you can quickly construct an elegant, reliable, and scalable Chat application. The interface of chat-uikit-uniapp is as demonstrated in the image below:



Supported Platform

Android
iOS
WeChat Mini Program
H5

Environment Requirements

HBuilderX (HBuilderX Version >= 3.8.4.20230531) or upgrade to the newest version

Vue2 / Vue3

Sass (sass-loader version ≤ 10.1.1)

Node (12.13.0 ≤ node version ≤ 17.0.0. The official LTS version 16.17.0 of Node.js is recommended.)

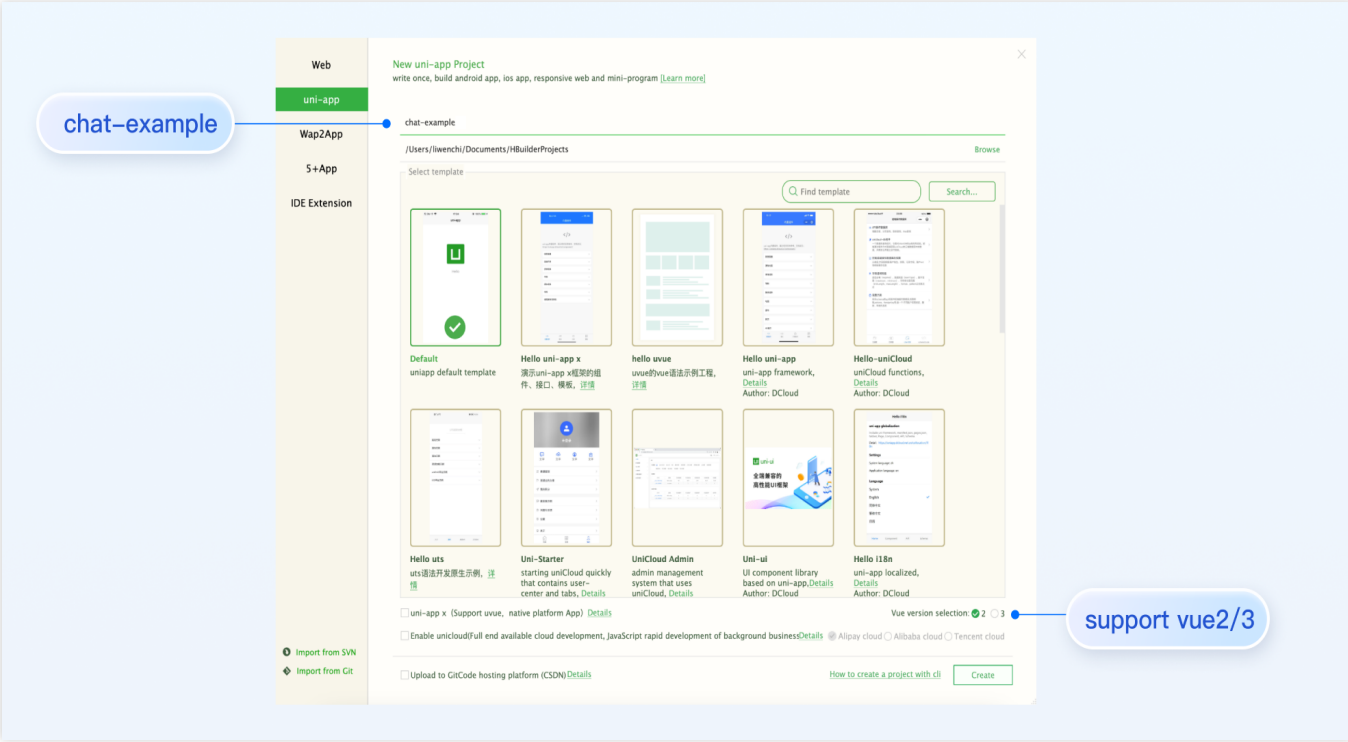
npm (use a version that matches the Node version in use)

TUIKit Source Code Integration

Follow the steps below to send your inaugural message.

Step 1: create a project (ignore this step if already has project)

Launch HbuilderX, select "File-New-Project" in the menu bar, and create a uni-app project named `chat-example` .



Step 2. Download the TUIKit component

Since HBuilderX does not create package.json files by default, you need to proactively create one. Execute the following command in the root directory of the project:

```
npm init -y
```

Download TUIKit and copy it to the source code:

macOS

Windows

Download the TUIKit component using the npm method:

```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

For ease of subsequent extensions, we propose that you replicate the TUIKit component to the pages directory within your project. Please conduct the following command in the root directory of your own project:

```
mkdir -p ./TUIKit && rsync -av --exclude=
{'node_modules','package.json','excluded-list.txt'}
./node_modules/@tencentcloud/chat-uikit-uniapp/ ./TUIKit
```

```
mkdir -p ./TUIKit/tui-customer-service-plugin && rsync -av
./node_modules/@tencentcloud/tui-customer-service-plugin/ ./TUIKit/tui-
customer-service-plugin
```

Download the TUIKit component using the npm method:

```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

For ease of subsequent extensions, we propose that you replicate the TUIKit component to the pages directory within your project. Please conduct the following command in the root directory of your own project:

```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-uniapp .\\TUIKit /i /e
/exclude:.\\node_modules\\@tencentcloud\\chat-uikit-uniapp\\excluded-list.txt
```

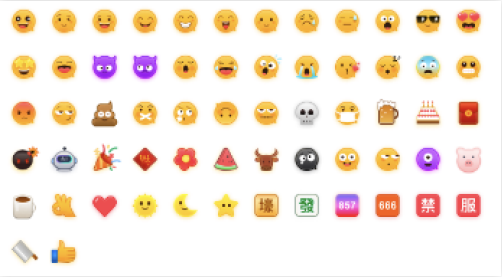
```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin
.\\TUIKit\\tui-customer-service-plugin /i /e
```

Step 3: Incorporate the TUIKit component

1. Project Configuration

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.



In the root directory, create vue.config.js (For Vue3 projects, please disregard this part).

```
const ScriptSetup = require('unplugin-vue2-script-setup/webpack').default;
module.exports = {
  parallel: false,
  configureWebpack: {
    plugins: [
      ScriptSetup({
        /* options */
      }),
    ],
  },
  chainWebpack(config) {
    // disable type check and let `vue-tsc` handles it
    config.plugins.delete('fork-ts-checker');
  },
};
```

Activate the split package configuration in the source code view of the `manifest.json` file

```
{
    "mp-weixin": {
        "appid": "",
        "optimization": {
            "subPackages": true
        }
    },
    "h5": {
        "optimization": {
            "treeShaking": {
                "enable": false
            }
        }
    }
}
```

2. Merge TUIKIt

Note:

Pursue the integration stringently in **Four Steps**. If you wish to package a Mini Program, please do not bypass the configuration of the "Home page of Mini Program Sub-package".

main.js file

pages.json file

App.vue file

Mini Program Sub-package Home Page

Pay heed, under Vue2 environment, make use of `Vue.use(VueCompositionAPI)` , to prevent inability to use environment variables such as `isPC` .

```
// Introduce the main package dependency
import TencentCloudChat  from "@tencentcloud/chat";
import TUICore from "@tencentcloud/tui-core";

import App from './App';

// #ifndef VUE3
import Vue from 'vue';
import './uni.promisify.adaptor';
import VueCompositionAPI from "@vue/composition-api";
Vue.use(VueCompositionAPI);
Vue.config.productionTip = false;
App.mpType = 'app';
const app = new Vue({
  ...App,
});
app.$mount();
// #endif

// #ifdef VUE3
import { createSSRApp } from 'vue';
export function createApp() {
  const app = createSSRApp(App);
  return {
    app,
  };
}
// #endif
```

```
{
 "pages": [{
  "path": "pages/index/index" // Your project's homepage
 }],
 "subPackages": [{
```

```
  "root": "TUIKit",
  "pages": [
   {
    "path": "components/TUIConversation/index",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
    }
   },
   {
    "path": "components/TUIChat/index",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
    }
   },
    // To integrate the chat component, this path must be configured: video playbac
   {
    "path": "components/TUIChat/video-play",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
    }
   },
   {
    "path": "components/TUIChat/web-view",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
    }
   },
   {
    "path": "components/TUIContact/index",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
    }
   },

   {
    "path": "components/TUIGroup/index",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
    }
   }
  ]
 }],
 "preloadRule": {
  "TUIKit/components/TUIConversation/index": {
   "network": "all",
   "packages": ["TUIKit"]
  }
```

```
 },
 "globalStyle": {
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "uni-app",
  "navigationBarBackgroundColor": "#F8F8F8",
  "backgroundColor": "#F8F8F8"
 }
}
```

```
<script lang="ts">
// #ifdef APP-PLUS || H5
import { TUIChatKit, genTestUserSig } from "./TUIKit";
import { vueVersion } from "./TUIKit/adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";
// #endif
// Mandatory information
const config = {
  userID: "test-user1", // User ID
  SDKAppID: 0, // Your SDKAppID
  secretKey: "", // Your secretKey
};
uni.$chat_userID = config.userID;
uni.$chat_SDKAppID = config.SDKAppID;
uni.$chat_secretKey = config.secretKey;

// #ifdef APP-PLUS || H5
uni.$chat_userSig = genTestUserSig(config).userSig;
// Initialization of TUIChatKit
TUIChatKit.init();
// #endif
export default {
  onLaunch: function () {
    // #ifdef APP-PLUS || H5
    // TUICore login
    TUILogin.login({
      SDKAppID: uni.$chat_SDKAppID,
      userID: uni.$chat_userID,
      // A UserSig is a cipher for users to sign in to Instant Messaging - it is es
      // This method is only suitable for running demos locally and debugging featu
      userSig: uni.$chat_userSig,
      // Should you require to transmit imagery, audio, video, files, and other for
      useUploadPlugin: true,
      // Local audit can identify and handle unsuitable and unsafe content to effec
      // This feature is an added service, please refer to: https://cloud.tencent.c
      // If you've purchased the content review service, please enable this feature
      useProfanityFilterPlugin: false,
```

```
      framework: `vue${vueVersion}` // Current development framework in use: vue2 /
    });
    // #endif
  },
 onShow: function() {
      console.log('App Show')
  },
 onHide: function() {
      console.log('App Hide')
  }
};
</script>
<style>
/*Common CSS for each page*/
uni-page-body,
html,
body,
page {
  width: 100% !important;
  height: 100% !important;
  overflow: hidden;
}
</style>
```

Note:

The mini program integrates by default in a subpackage. The login must be completed on the TUIKit startup page.

If you do not require the packaging of mini-programs (for instance, building H5 only), you can disregard the configuration content of "*Mini Program Split Package Homepage*".

Example: The TUIKit sub-package first screen launch page is the **TUIConversation** page

Step 1: Create a subPackage-init.ts file under the TUIKit/components/TUIConversation directory

```
import { TUIChatKit, genTestUserSig } from "../../index.ts";
import { vueVersion, onMounted } from "../../adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";

// Initialization of TUIChatKit
TUIChatKit.init();
uni.$chat_userSig = genTestUserSig({
        userID: uni.$chat_userID,
        SDKAppID: uni.$chat_SDKAppID,
        secretKey: uni.$chat_secretKey
}).userSig;

// login
TUILogin.login({
```

```
    SDKAppID: uni.$chat_SDKAppID,
    userID: uni.$chat_userID,
    // UserSig is the cipher for users to sign in to Instant Messaging, essentially b
    // This method is only suitable for running Demo locally and debugging functions.
    userSig: uni.$chat_userSig,
    // Should you require to send image, voice, video, file and other rich media mess
    useUploadPlugin: true,
    // Local review can successfully identify and handle inappropriate and unsafe con
    // This functionality is a value-added service, please refer to: https://cloud.te
    // If you have purchased the content review service, to activate this feature ple
    useProfanityFilterPlugin: false,
    framework: `vue${vueVersion}` // Current development uses framework vue2 / vue3
  }).then(() => {
    uni.showToast({
      title: "login success"
    });
  });
```

Step 2: Import within TUIKit/components/TUIConversation/index.vue

```
// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

See the following figure:

```
<script lang="ts" setup>
import {
  TUIStore,
  TUIGlobal,
  StoreName,
} from "@tencentcloud/chat-uikit-engine";
import { ref } from "../../adapter-vue";
import ConversationList from "./conversation-list/index.vue";
import ConversationHeader from "./conversation-header/index.vue";
import ConversationNetwork from "./conversation-network/index.vue";
import { onHide } from "@dcloudio/uni-app"; // 该方法只能用在父组件内，子组件内不生效

// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

3. Configuring the entry points of TUIConversation and TUIContact on the main package homepage of the project

Create an index.vue file under the pages/index folder

```
<template>
  <div class="index">
        <p class="index-button" @click="openConversation">Open TUIKit Conversation<
        <p class="index-button" @click="openContact">Open TUIKit Contacts</p>
  </div>
</template>
<script>
export default {
methods: {
  // Open the TUIKit session list
  openConversation() {
    uni.navigateTo({
          url: "/TUIKit/components/TUIConversation/index",
        });
  },
  // Accessing TUIKit Contacts
  openContact() {
    uni.navigateTo({
          url: "/TUIKit/components/TUIContact/index",
        });
  },
 },
};
</script>
<style lang="scss" scoped>
.index {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  &-button {
    width: 180px;
        padding: 10px 40px;
        color: #fff;
        background-color: #006eff;
        font-size: 16px;
        margin-top: 65px;
        border-radius: 30px;
        text-align: center;
  }
}
</style>
```

Step 4: Gain access to SDKAppID, secretKey, and userID

Within the App.vue file in the root directory of configuration, find `config` object's SDKAppID, secretKey, and userID. The SDKAppID and secretKey can be accessed through the Instant Messaging Console, and the userID can be accessed when creating an account in the Instant Messaging Console.

```
// Mandatory information
const config = {
  userID: "test-user1", // Login User ID
  SDKAppID: 0, // Your SDKAppID
  secretKey: "", // Your secretKey
};
```

access SDKAppID,secretKey

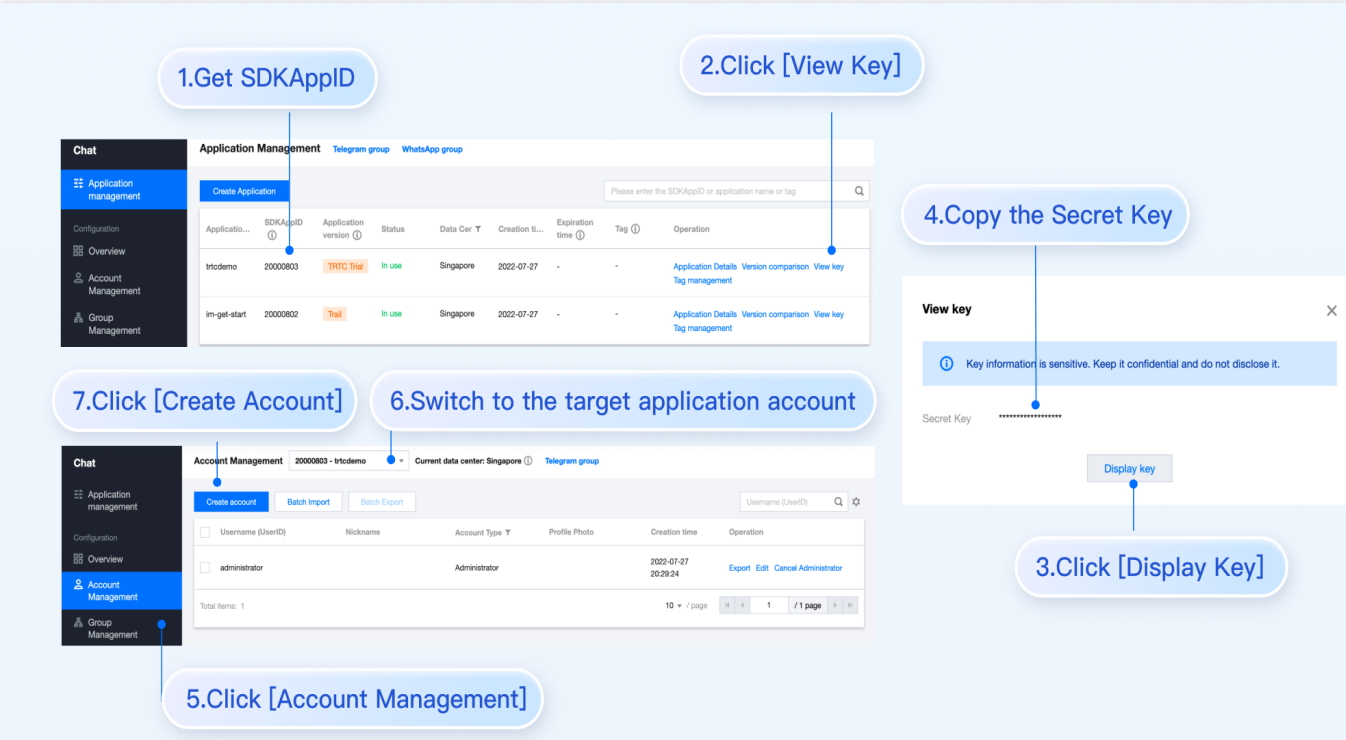
In the Instant Messaging Console under the **application management** page, you can see the applications you have created. The SDKAppID is in the second column. Then click on the **peekKey** in the operation options. A dialogue box will appear on the website for the peekKey, and by clicking on the **Show Key**, the peekKey will be revealed.

Create an account with `userID` as `test-user1`

Click on **Account Management** on the left side of the console. If you have multiple applications, ensure to switch to your current application. Then, under the current application, click **Create new account** to create an account with a userID of `test-user1` .

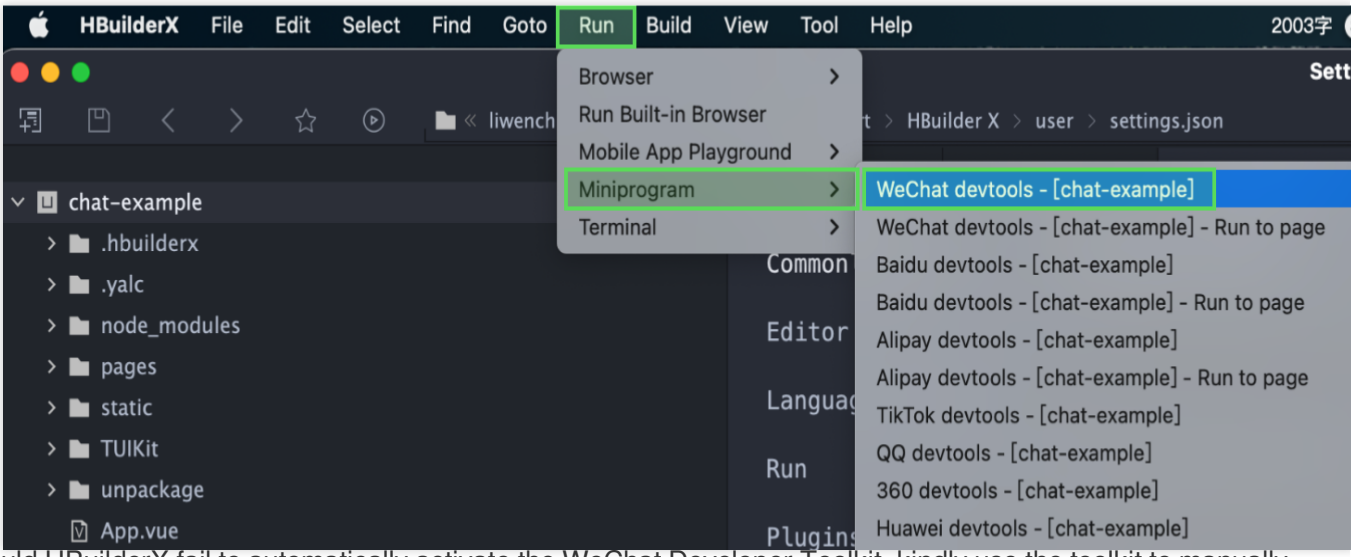
Note:

The step of creating an account can be circumvented as TUIKit will auto-generate an account during the sign in process if the configuration's userID does not exist. This only demonstrates how to access the userID.



Step 5. Launch the project

1. Launch the project using HBuilderX, then click on "Run - Run to Mini Program Simulator - WeChat Developer Tools".



2. Should HBuilderX fail to automatically activate the WeChat Developer Toolkit, kindly use the toolkit to manually open the compiled project.

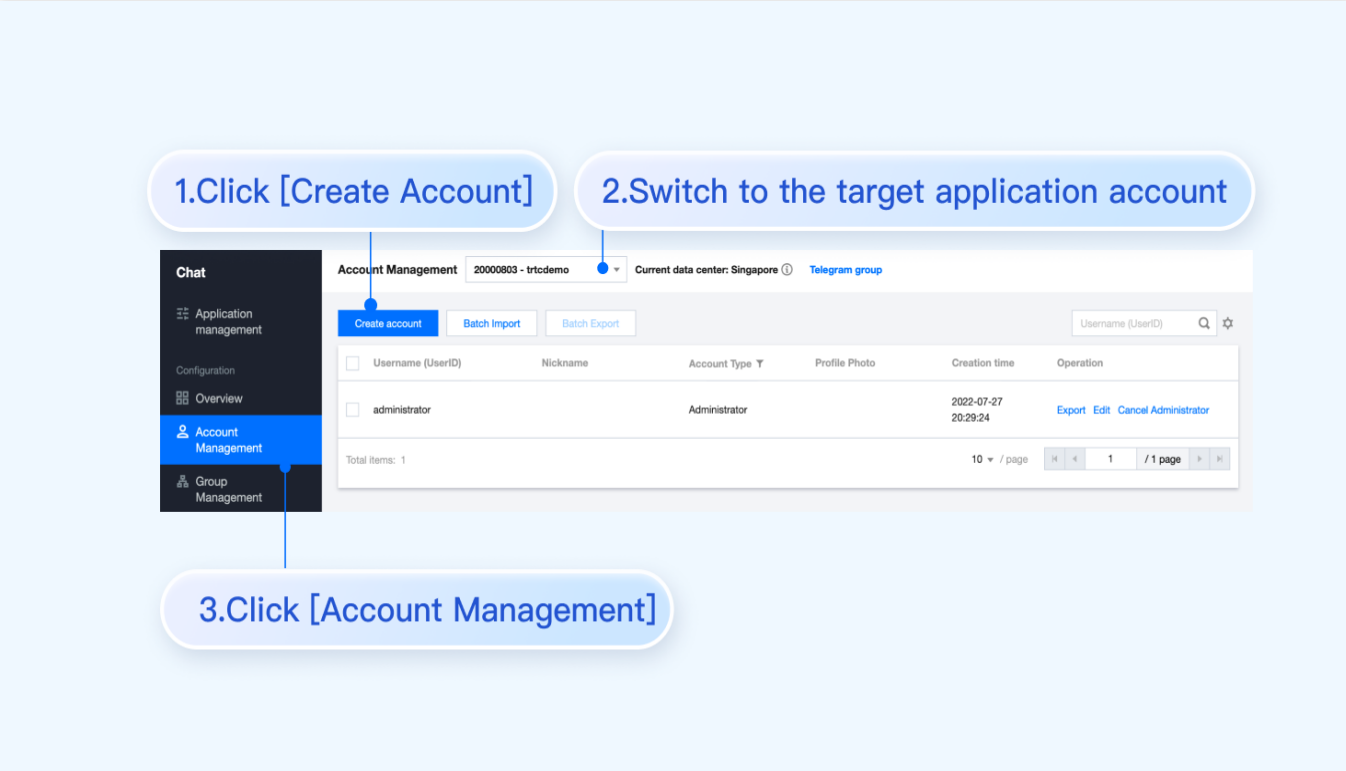
Open the `unpackage/dist/dev/mp-weixin` under the project root directory using the WeChat Developer Tool.

3. After opening the project, check the "Do not verify valid domain, web-view (business domain), TLS version, and HTTPS certificate" in "Details-Local Setting" of WeChat Developer Tools.

Step 6. Send your first message

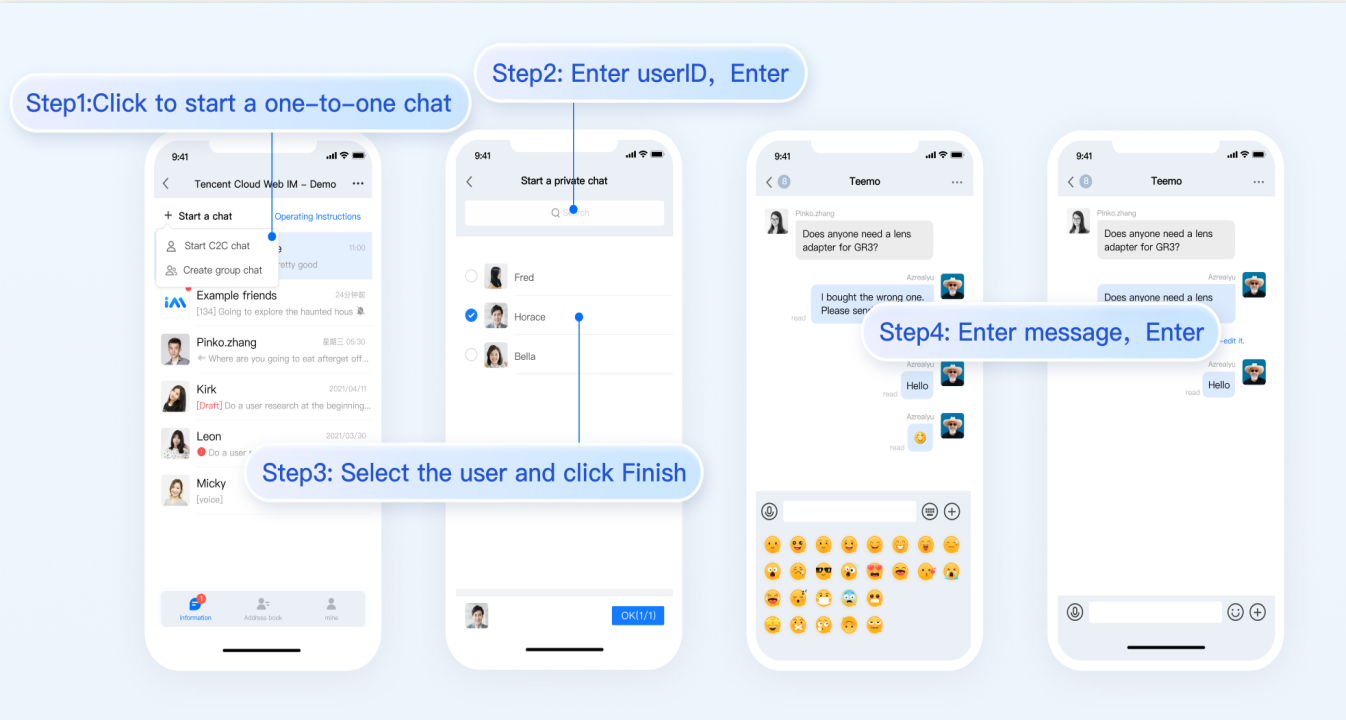
1. Create a User account through the Instant Messaging Console

Navigate to the **Account Management** page from the left sidebar, and click on **New Account** to create a regular account with userID:test-user2.



2. Run project and create conversation.

click **to open TUIKit conversation** , search for user userID:test-user2, and send your first message.



Additional Advanced Features

Audio-Visual Communication TUICallKit Plugin

Note:

The TUICallKit audio/video component is not integrated by default in TUIKit,TUICallKit primarily handles voice and video calls.

Should you need to integrate call functionalities, kindly refer to the following documents for guidelines.

For packaging into APP, refer to: Audio/Video Calling (Client)

For packaging to Miniprogram, please refer to: Video Calls(Miniprogram)

For packaging into HTML5, please refer to the official documentation: Audio and Video Calls (HTML5)

Please stay tuned.

TIMPush Offline Push Plugin

Indication

By default, TUIKit does not integrate the TIMPush **offline push plugin**.TIMPush is Tencent Cloud's Instant Messaging Push Plugin. Currently, offline push supports Android and iOS platforms, and devices include: Huawei, Xiaomi, OPPO, Vivo, Meizu, and Apple phones.

Should you require the integration of offline push capabilities within your APP, kindly refer to the implementation of uni-app offline push.

Please stay tuned.

Individually integrate TUIChat component

Consider Independent Integration of TUIChat Component as a Solution

FAQs

For additional inquiries, please refer to Uniapp FAQ.

Exchange and Feedback

Click here to join the IM community, where you'll receive support from experienced engineers to help overcome your challenges.

Reference Documentation

Related to UIKit (vue2 / vue3):

Source code of chat-uikit-uniapp (vue2/vue3) on GitHub

Rapid Incorporation of chat-uikit-uniapp npm

Regarding ChatEngine:

ChatEngine API Manual

ChatEngine npm

Vue

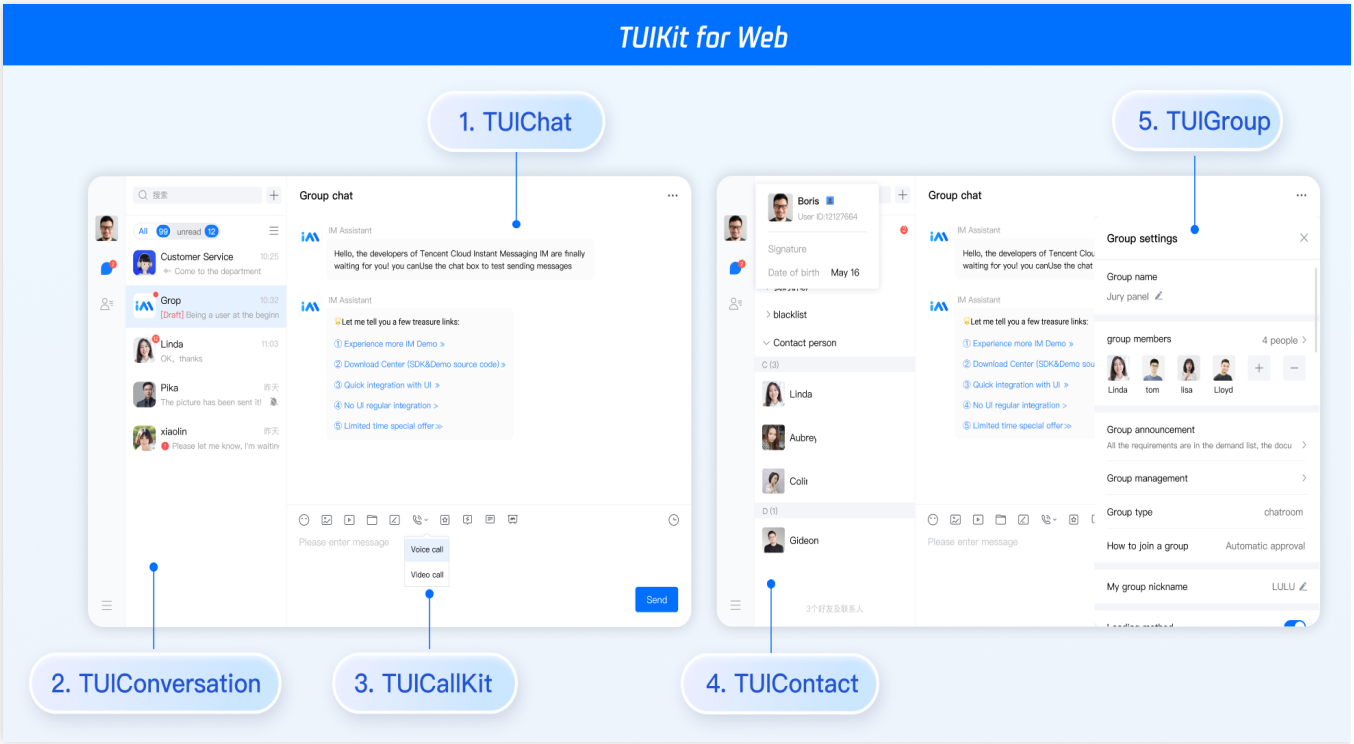
Last updated：2024-11-04 16:57:57

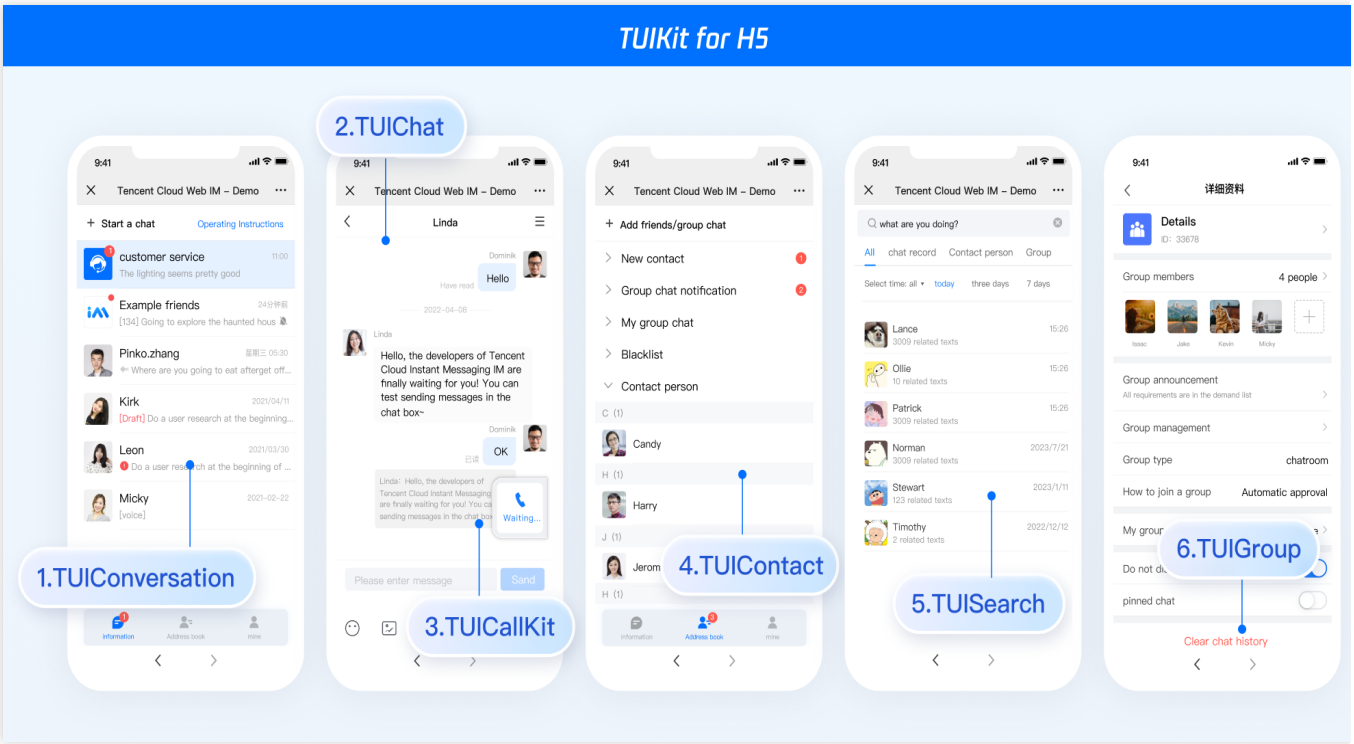
TUIKit Overview

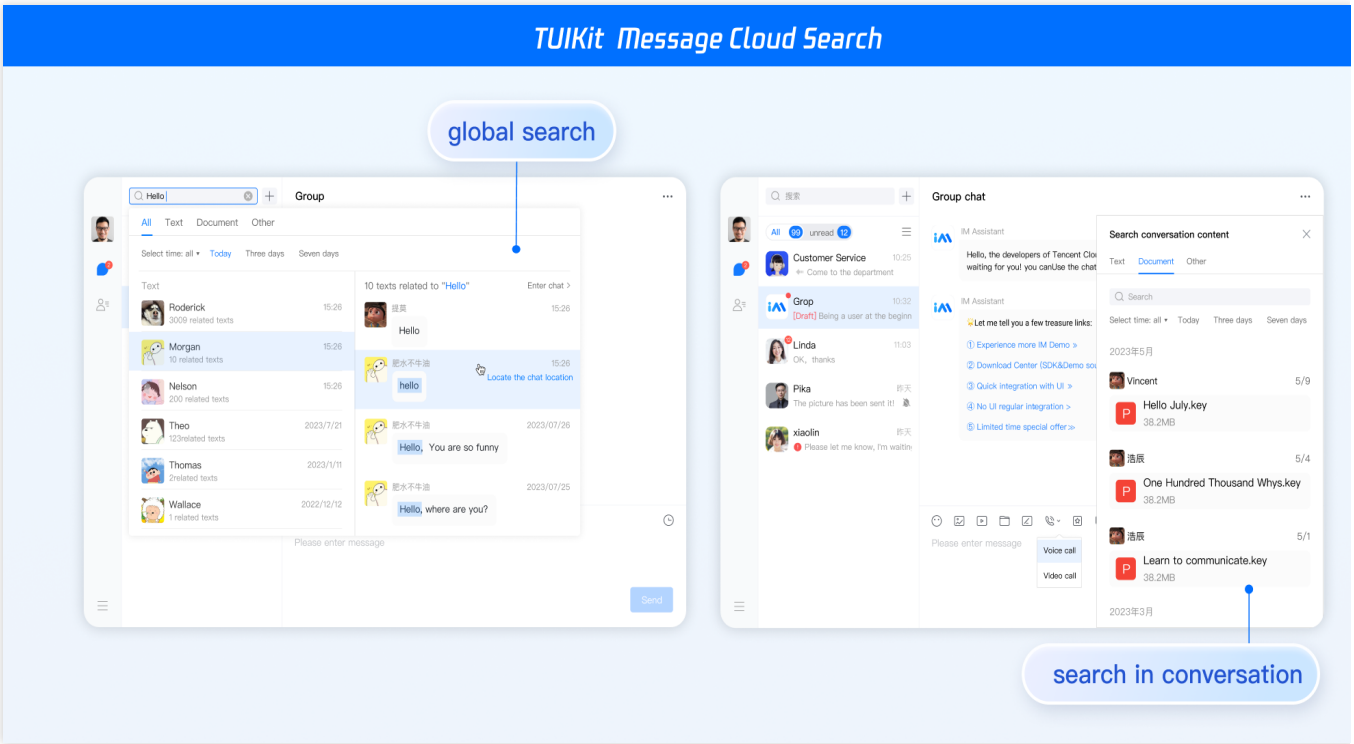
TUIKit is a UI component library based on Tencent Cloud Chat SDK. It provides some universal UI components, including conversations, chats, relationship chains, groups, audio/video calls, and other features.
With these UI components, you can quickly build your own in-app chat.
While implementing UI features, the components in TUIKit will call the corresponding interfaces of the Chat SDK to implement Chat-related logic and data processing. Therefore, developers only need to focus on their own business or personalized expansion when using TUIKit.

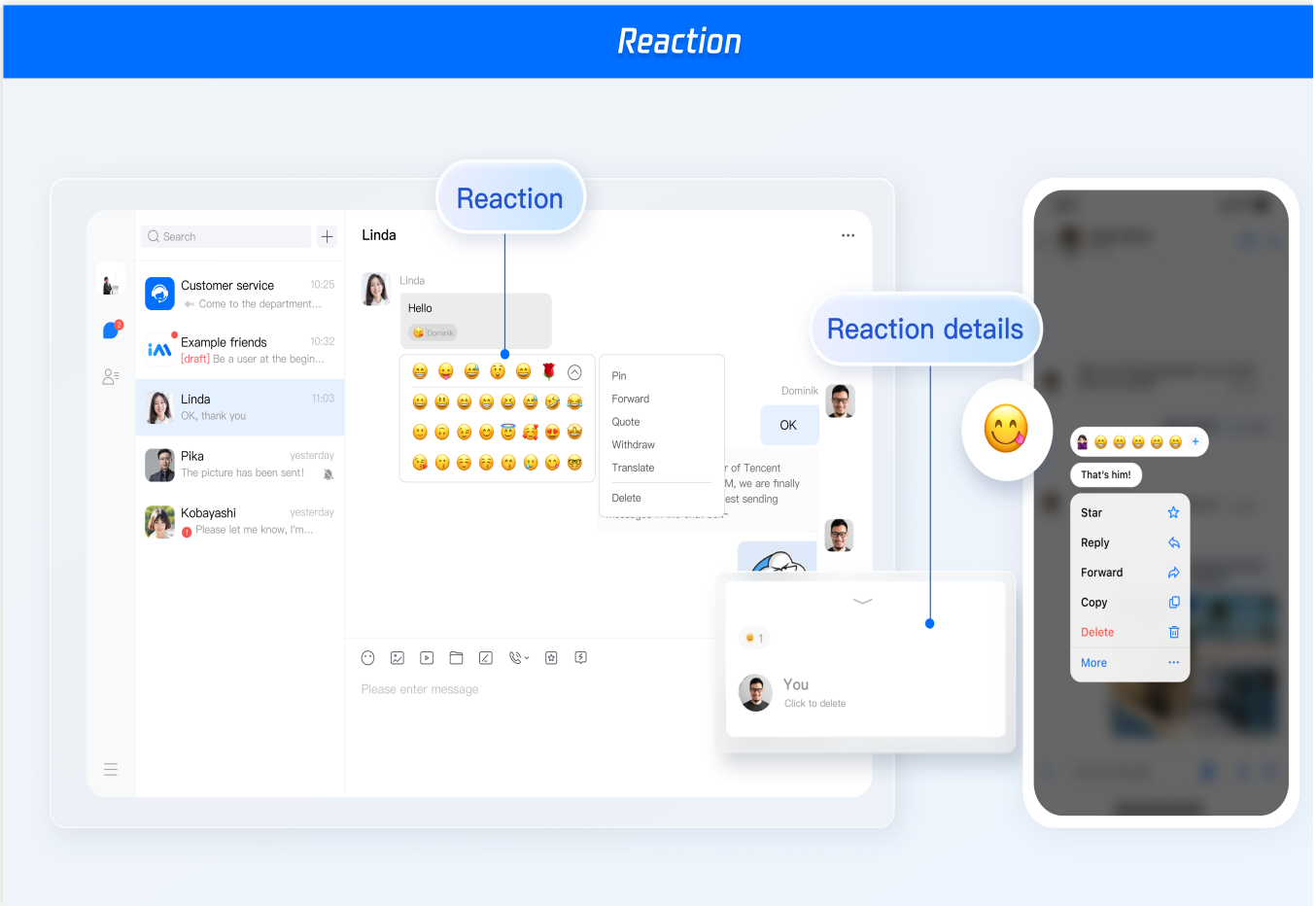
TUIKit Components

TUIKit is mainly divided into several UI sub-components: TUIChat, TUIConversation, TUIGroup, TUIContact, and TUISearch.
Each UI component is responsible for displaying different content.




TUIKit for H5
1.TUIConversation
2.TUIChat
3.TUICallKit
4.TUIContact
5.TUISearch
6.TUIGroup


TUIKit Message Cloud Search
global search
search in conversation



Environment Requirements

Vue (Fully compatible with both Vue2 & Vue3. While incorporating below, please select the Vue version guide that matches your needs)

TypeScript (Should your project be based on JavaScript, please proceed to JS project integrate to set up a progressive support for TypeScript)

Sass (sass-loader ≤ 10.1.1)

node(node.js ≥ 16.0.0)

npm (use a version that matches the Node version in use)

Integration of TUIKit (Web & H5)

Step 1. Create a project

TUIKit supports creating a project structure using webpack or vite, configured with Vue3 / Vue2 + TypeScript + sass. Below are a few examples of how to construct your project:

vue-cli

vite

Please Note:

Please make sure you have **@vue/cli version 5.0.0 or above** . The following sample code can be used to upgrade your @vue/cli version to v5.0.8.

Establish a project using Vue CLI, with configuration set to Vue2/Vue3 + TypeScript + Sass/SCSS.

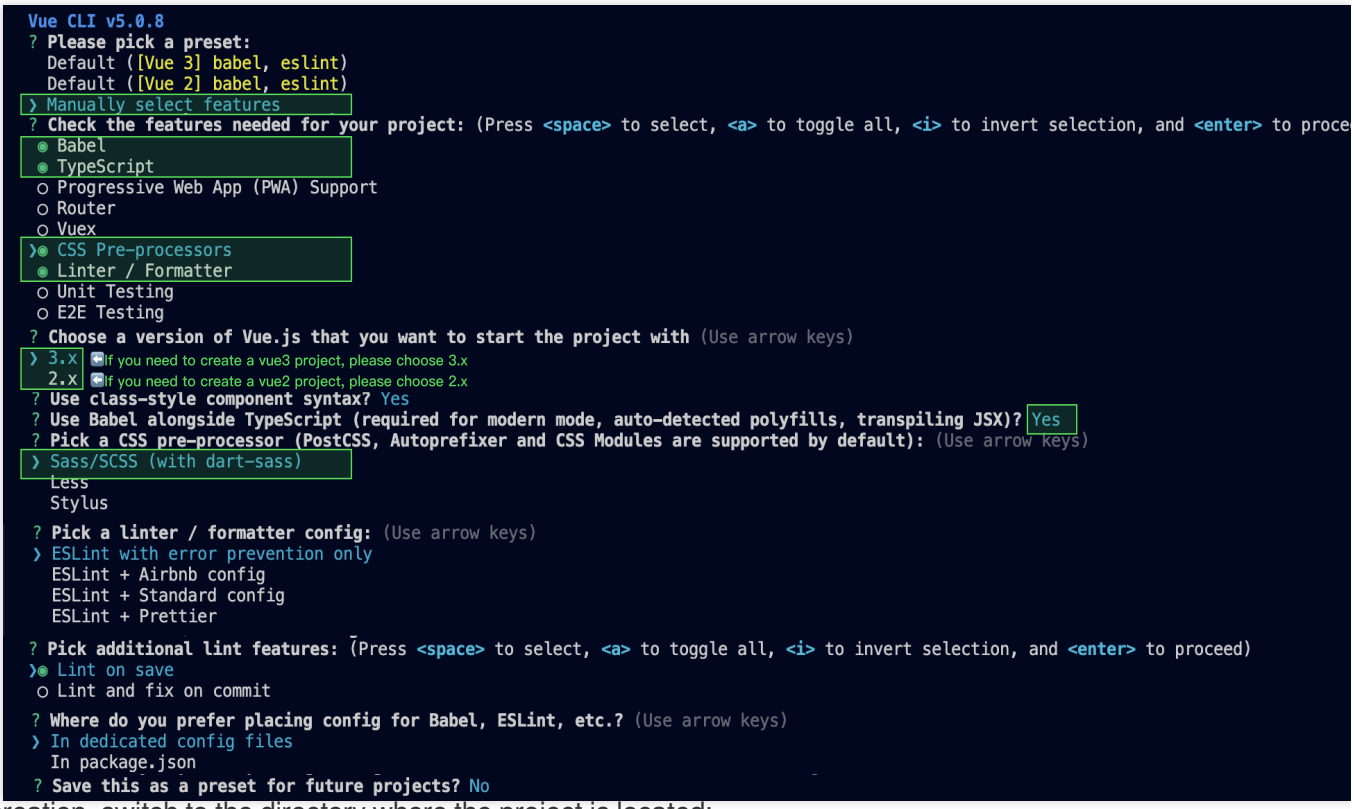
If Vue CLI is not yet installed, or the version is below 5.0.0, you can use the following method for installation via Terminal or CMD:

```
npm install -g @vue/cli@5.0.8 sass sass-loader@10.1.1
```

Create a project through Vue CLI and select the configuration items depicted below.

```
vue create chat-example
```

Please make sure to select according to the following configuration:



After creation, switch to the directory where the project is located:

```
cd chat-example
```

If you are a vue2 project, please make the following corresponding environment configurations based on the Vue version you are using.

IIf you are a vue2 project, please ignore.

vue2.7

Vue 2.6 and below

```
npm i vue@2.7.9 vue-template-compiler@2.7.9
```

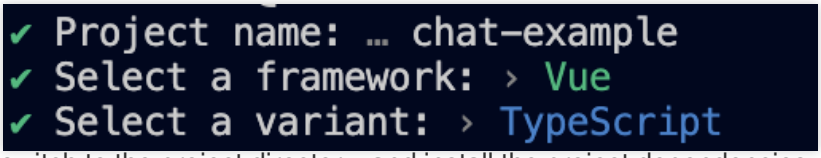
```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compiler@2.6.14
```

Please Note:

Vite requires **Node.js versions 18+, 20+.** Pay attention to upgrade your Node version when your package manager issues a warning, for more details refer to Vite official website.

Create a project using Vite, configure Vue + TypeScript according to the options in the picture below.

```
npm create vite@latest
```



Then, switch to the project directory, and install the project dependencies:

```
cd chat-example
npm install
```

Install the sass environment dependency required for TUIKit:

```
npm i -D sass sass-loader
```

Step 2. Download the TUIKit component

Download the TUIKit component through npm. To facilitate your subsequent expansion, it is recommended that you copy the TUIKit component to the src directory of your project:

macOS

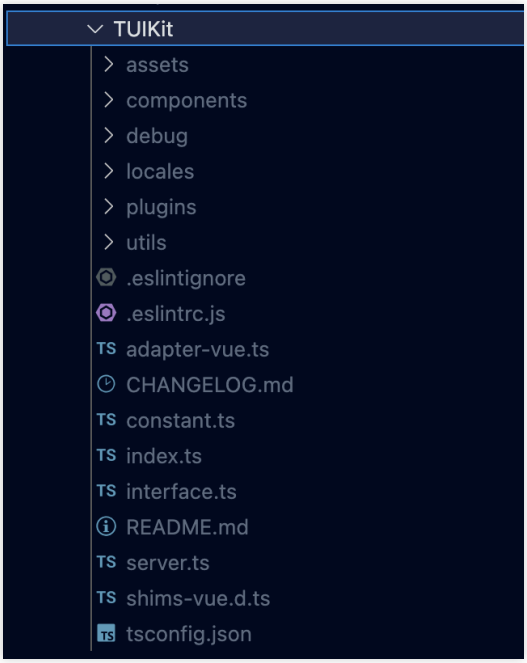
Windows

```
npm i @tencentcloud/chat-uikit-vue
mkdir -p ./src/TUIKit && rsync -av --exclude=
{'node_modules','package.json','excluded-list.txt'}
./node_modules/@tencentcloud/chat-uikit-vue/ ./src/TUIKit
```

```
npm i @tencentcloud/chat-uikit-vue
```

```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-vue .\\src\\TUIKit /i /e
/exclude:.\\node_modules\\@tencentcloud\\chat-uikit-vue\\excluded-list.txt
```

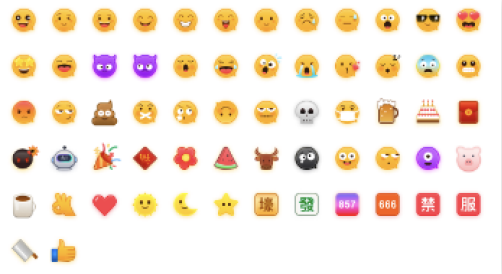
Upon successful completion, the directory structure is depicted as follows:



Step 3. Import TUIKit component

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.



On the page where you want to display it, simply import the TUIKit component to use it.

For example, implementing the following code on the App.vue page allows for a quick setup of the chat interface (the following example code supports both Web and H5):

Note:

The example code below uses the setup syntax. If your project does not use the setup syntax, please register components according to the standard methods of Vue3/Vue2.

vue3

vue2.7

vue2.6 and below

```
<template>
  <div id="app">
    <TUIKit :SDKAppID="0" userID="xxx" userSig="xxx" />
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue';
</script>
<style lang="scss">
</style>
```

```
<template>
  <div id="app">
    <TUIKit :SDKAppID="0" userID="xxx" userSig="xxx" />
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue2';
</script>
<style lang="scss">
</style>
```

```
<template>
  <div id="app">
    <TUIKit :SDKAppID="0" userID="xxx" userSig="xxx" />
    <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullScre
  </div>
</template>
<script lang="ts" setup>
import { TUIKit } from './TUIKit';
import { TUICallKit } from '@tencentcloud/call-uikit-vue2.6';
</script>
<style lang="scss">
</style>
```

1. Install dependencies supporting composition-api and script setup, as well as dependencies related to vue2.6.

```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

2. Import VueCompositionAPI in `main.ts/main.js` .

```
import VueCompositionAPI from "@vue/composition-api";
Vue.use(VueCompositionAPI);
```

3. Add the following in `vue.config.js` . If the file does not exist, please create it.

```
const ScriptSetup = require("unplugin-vue2-script-setup/webpack").default;
module.exports = {
  parallel: false, // disable thread-loader, which is not compactible with this plu
  configureWebpack: {
    plugins: [
      ScriptSetup({
        /* options */
      }),
    ],
  },
  chainWebpack(config) {
    // disable type check and let `vue-tsc` handles it
    config.plugins.delete("fork-ts-checker");
  },
};
```

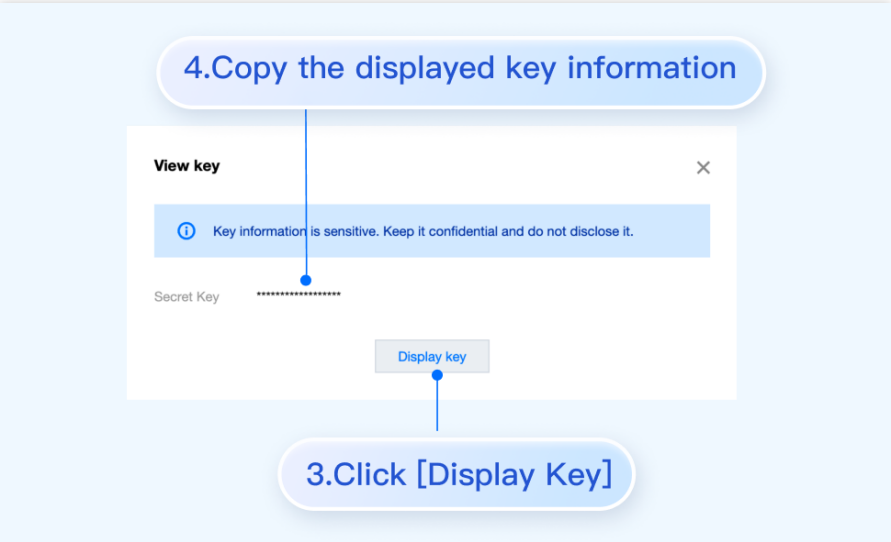
4. At the end of the `src/TUIKit/adapter-vue.ts` file, replace the export source:

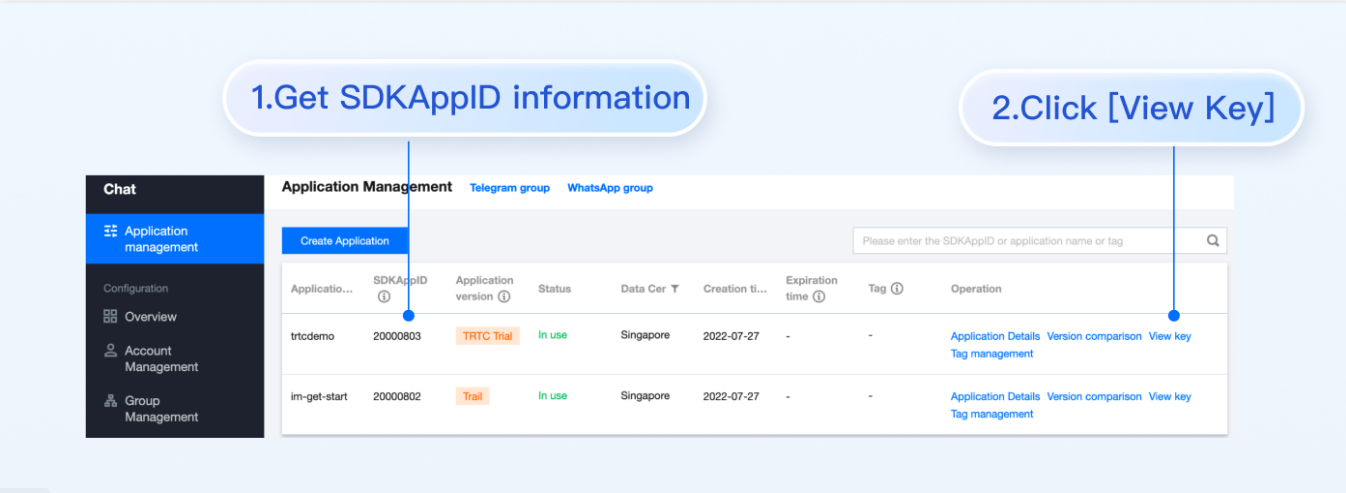
```
// Initial notation
export * from "vue";
// Replace with
export * from "@vue/composition-api";
```

Step 4: Secure SDKAppID, secretKey, and userID

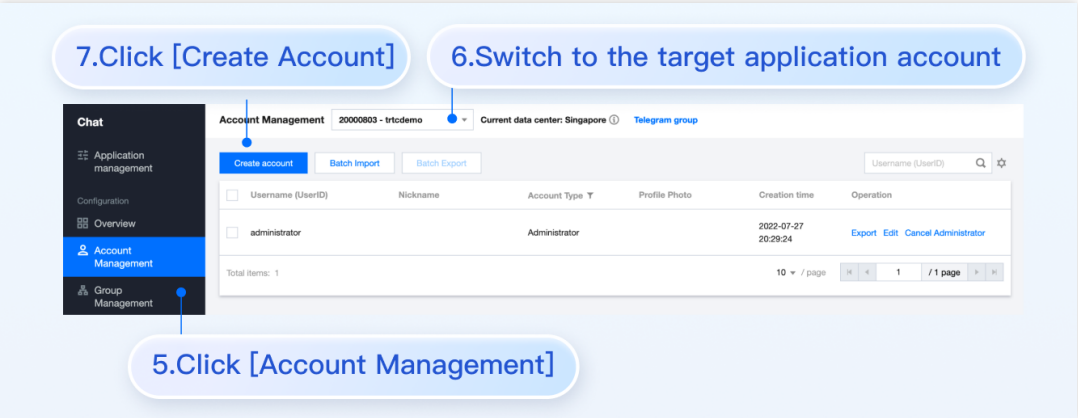
Set the relevant parameters `SDKAppID` , `secretKey` , and `userID` in the example code of the `main.ts / main.js` file:

`SDKAppID` and `SecretKey` can be accessed by the Chat Console:





`UserID` can be accessed by the Chat Console > Account Management. Switch to the target application account to create an account and get the userID.



Step 5. Launch the project

vue-cli

vite

```
npm run serve
```

```
npm run dev
```

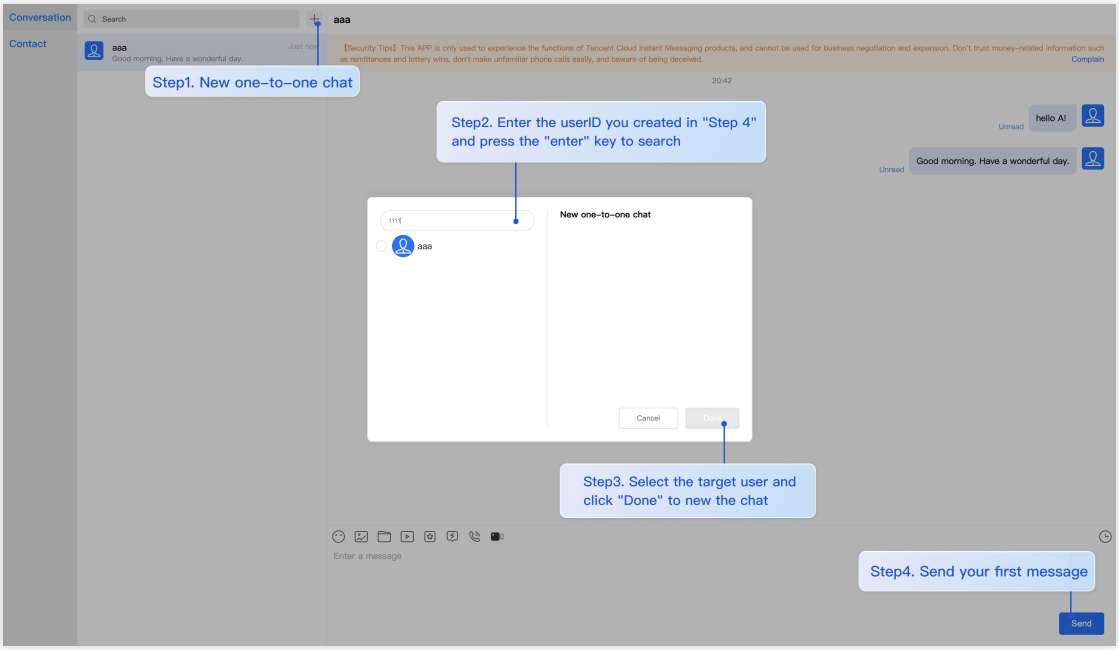
Additional item: Switching languages

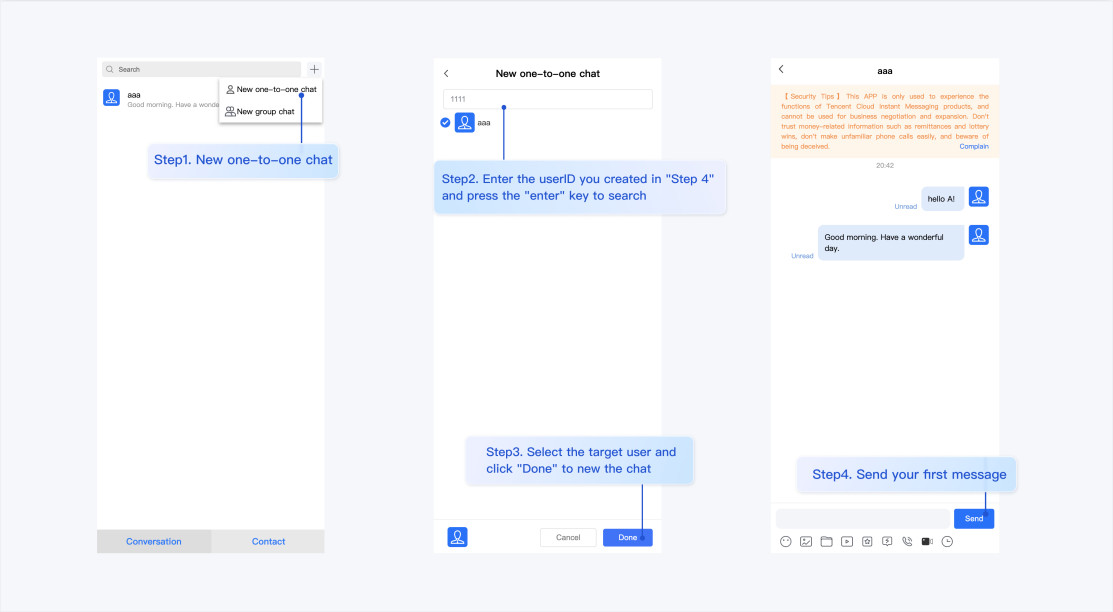
The `Vue TUIKit` comes with default **Simplified Chinese, English** language packages that serve as the interface display language.

You may switch languages through the following methods. For more methods, see Internationalization Interface Language.

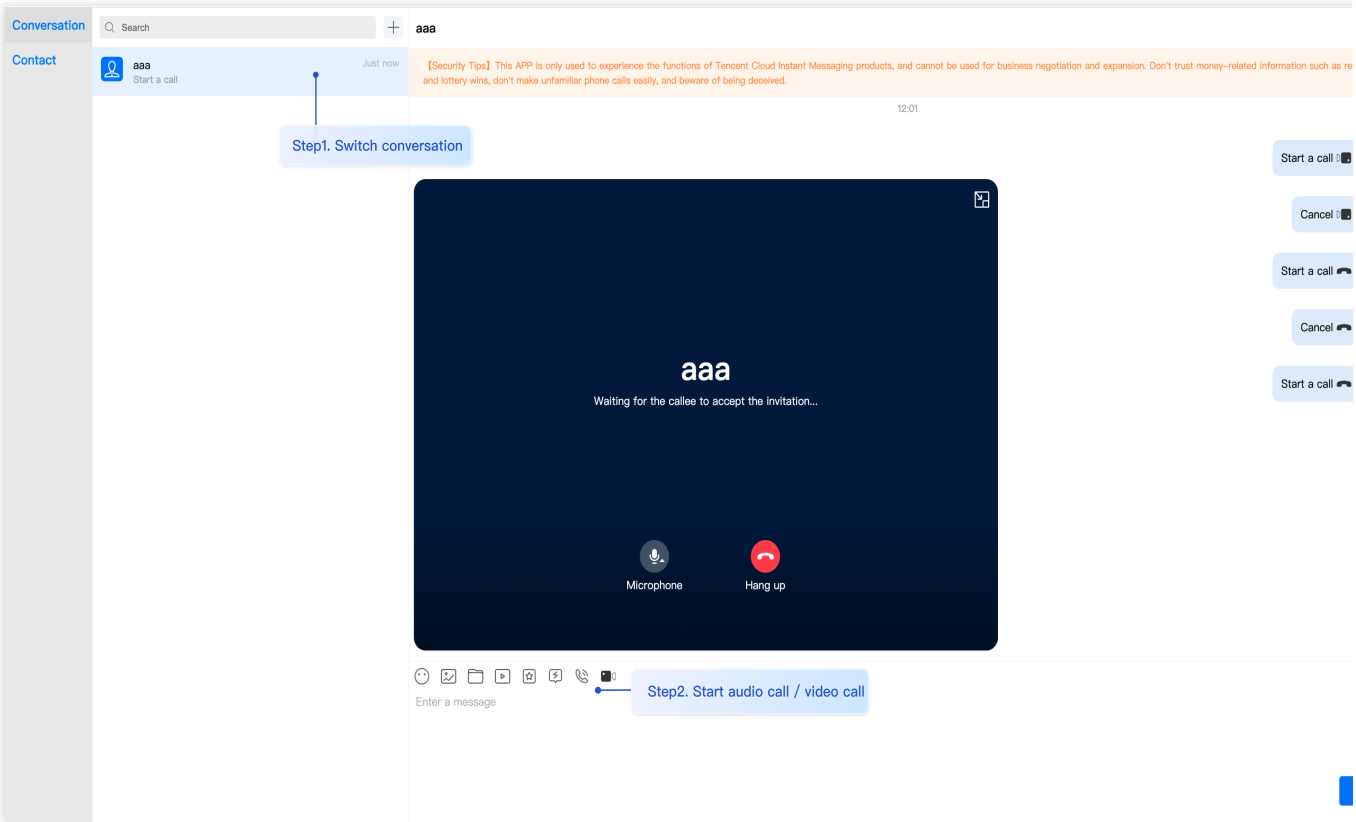
```
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
// change language to chinese
TUITranslateService.changeLanguage("zh");
// change language to english
TUITranslateService.changeLanguage("en");
```

Step 6. Send your first message





Step 7: Make your first phone call



FAQs

Product Service FAQs

1. The Audio/Video Call Capability package is not activated? Failure to initiate the Audio/Video Call?

Please click Audio/Video Call > Frequently Asked Questions to view the solutions.

2. What is UserSig? How is UserSig generated?

A UserSig is a password with which you can log in to use IM service. It is the ciphertext generated by encrypting information such as userID.
The issuance of UserSig is achieved by integrating the calculation code for UserSig into your server-side, whilst providing an interface designed for your project. Whenever UserSig is required, your project could request the operational server for a dynamic UserSig. For further information, please refer to Generating UserSig on the server-side.

Caution

The method to obtain UserSig demonstrated in this document utilizes the configuration of a SECRETKEY within the client-side code. Within this procedure, the SECRETKEY is notably vulnerable to decompilation and reverse-engineering. Should your SECRETKEY be leaked, malefactors could potentially exploit your Tencent Cloud traffic. Therefore, **this technique is only appropriate for local operation and functional debugging**. For the correct method of issuing UserSig, please refer to the earlier text.

Connection Errors FAQs

1. Runtime error: "TypeError: Cannot read properties of undefined (reading "getFriendList")"

If the following errors occur during runtime after connecting as per the steps outlined above, it is imperative that you **delete the node_modules directory under the TUIKit folder** to ensure the uniqueness of TUIKit's dependencies, preventing issues caused by multiple copies of dependencies.

```
▼[Vue warn]: Error in v-on    vue.runtime.esm.js:4568
handler (Promise/async): "TypeError: Cannot read
properties of undefined (reading 'getFriendList')"

found in

---> <Index> at
src/TUIKit/components/TUISearch/index.vue
       <App> at src/App.vue
         <Root>
 warn @ vue.runtime.esm.js:4568
```

2. How does a JS project integrate the TUIKit component?

TUIKit exclusively supports the TS environment for operation. You can enable the coexistence of existing JS code in your project with the TS code in TUIKit through progressive configuration of TypeScript.
vue-cli
vite
Please execute the following in the root directory of your engineering project created by the Vue CLI scaffold:

```
vue add typescript
```

Subsequently, please make selections in accordance with the following configuration options. To assure that we can support both the existing js code and the ts code within TUIKit, it is imperative that you strictly adhere to the five

options presented below.

```
Run `npm audit` for details.
✔  Successfully installed plugin: @vue/cli-plugin-typescript

? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)?
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? Yes


🚀  Invoking generator for @vue/cli-plugin-typescript...
📦  Installing additional dependencies...
```

Once these steps are completed, please rerun the project!

Please execute the following command in your project's root directory created with vite:

```
npm install -D typescript
```

3. Runtime error reported: /chat-example/src/TUIKit/components/TUIChat/message-input/message-input-editor.vue .ts(8,23)TS1005: expected.

```
98% after emitting CopyPlugin

ERROR  Failed to compile with 2 errors                                    16:20:59

error  in [redacted]/chat-example/src/TUIKit/components/TUIChat/message-input/message-input-editor.vue.ts

[tsl] ERROR in [redacted]/chat-example/src/TUIKit/components/TUIChat/message-input/message-input-editor.vue.ts(
3)
      TS1005: ',' expected.
```

The above error message appears because your installed @vue/cli version is too low. **You must ensure that your @vue/cli version is 5.0.0 or higher**. The upgrade method is as follows:

```
npm install -g @vue/cli@5.0.8
```

4. Runtime error: Failed to resolve loader: sass-loader

```
ERROR  Failed to compile with 1 error

Failed to resolve loader: sass-loader
You may need to install it.
No issues found.
```

The above error message appears because the `sass` environment is not installed on your machine, please run the following command to install the `sass` environment:

```
npm i -D sass sass-loader@10.1.1
```

5. Other ESLint errors?

If copying chat-uikit-vue to the src directory results in error due to inconsistency with your local project code style, you may ignore this component directory. This can be achieved by adding .eslintignore file to the project root directory:

```
# .eslintignore
src/TUIKit
```

6. How to disable the full screen overlay error message prompt of webpack in dev mode in vue/cli?

You can disable it in the vue.config.js file at the root directory of your project:

webpack4

webpack3

```
module.exports = defineConfig({
  ...
  devServer: {
    client: {
      overlay: false,
    },
  },
});
```

```
module.exports = {
  ...
  devServer: {
    overlay: false,
  },
};
```

7. What to do when encountering 'Component name "XXXX" should always be multi-word'?

The version of ESLint utilized in IM TUIKit web is v6.7.2, which does not rigidly verify the camelCase format for module names.

Should you encounter this dilemma, you may configure as follows in the .eslintrc.js file:

```
  module.exports = {
    ...
    rules: {
        ...
        'vue/multi-word-component-names': 'warn',
    },
  };
```

8. What should I do if I encounter ERESOLVE unable to resolve dependency tree?

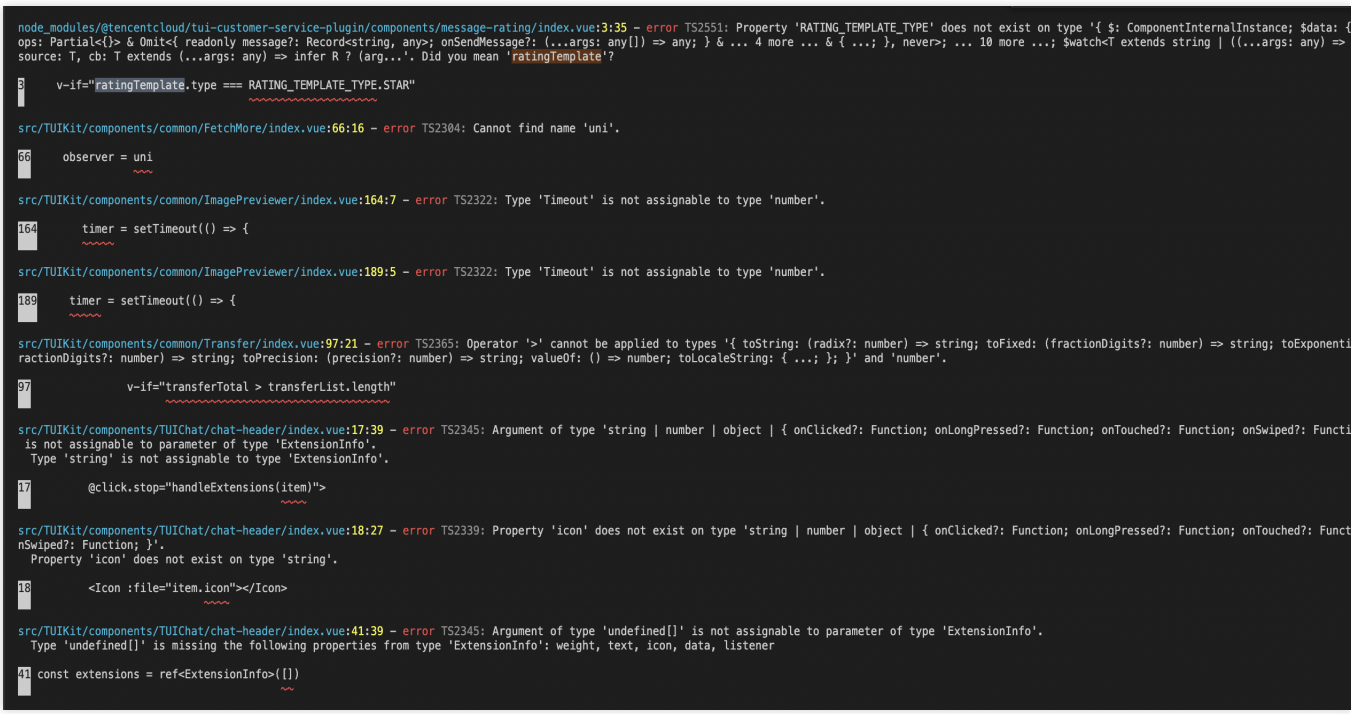
If ERESOLVE unable to resolve dependency tree appears when npm install is run, it indicates a conflict in dependency installation. The following method can be adopted for installation:

```
npm install --legacy-peer-deps
```

9. How might one address the error message, 'vue packages version mismatch' occurring during execution?

```
// If you are using a vue2.7 project, please execute in your project root directory
npm i vue@2.7.9 vue-template-compiler@2.7.9
// If you have a Vue2.6 project, please execute in your project's root directory
npm i vue@2.6.14 vue-template-compiler@2.6.14
```

10. Why does TypeScript report an error after npm run build in a Vite project?



Reason: It's led by the vue-tsc command in "build": "vue-tsc && vite build" under package.json script.

```
"scripts": {
  "dev": "vite",
  "build": "vue-tsc && vite build",
  "preview": "vite preview"
},
```

Solution: Simply remove vue-tsc. "build": "vite build"

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
},
```

Contact Us

Join the Telegram technical discussion group or WhatsApp discussion group, enjoy the support of professional engineers, and solve your difficulties.

Documentation

Related to Vue2 & Vue3 UIKit:

chat-uikit-vue npm
Vue2 Demo Source Code and Running Example
Vue3 Demo Source Code and Running Example

Vue2 & Vue3 UIKit logic layer: engine

chat-uikit-engine npm
chat-uikit-engine interface

Unity

Last updated：2024-11-04 16:57:57

This document describes how to integrate the SDK for Unity.

Environment Requirements

Environment	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.

Supported Platforms

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

Platform	Chat SDK
iOS	Supported
Android	Supported
macOS	Supported
Windows	Supported
Web	Supported from 1.8.1+

Note

For web, you need to perform a few extra steps for SDK integration. For details, see Part 5.

Prerequisites

1. You have signed up for a Tencent Cloud account and completed identity verification.

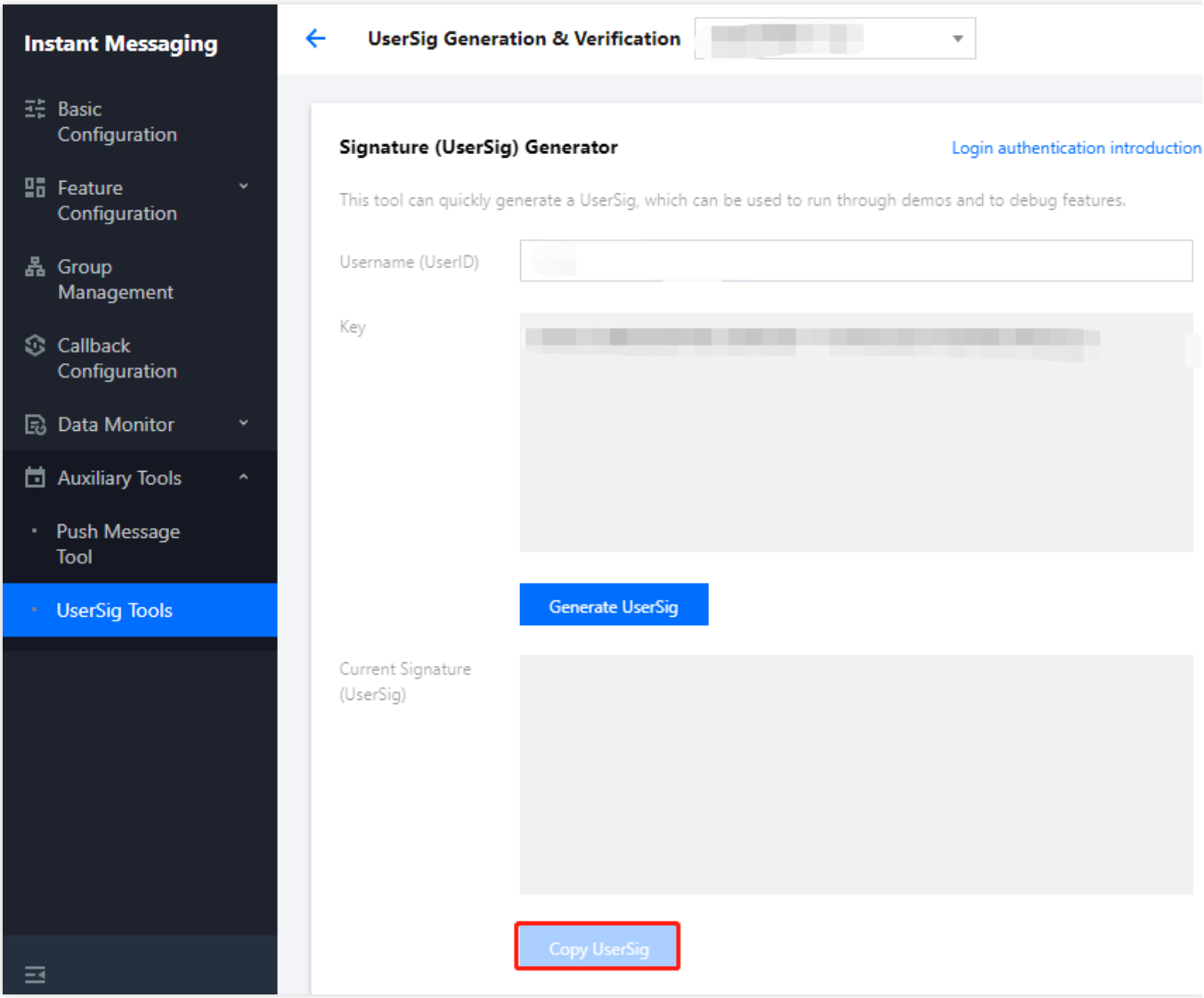
2. You have created an application as instructed in Creating and Upgrading an Application and recorded the SDKAppID.

Part 1. Creating Test Accounts

In the Chat console, select your application and click **Auxiliary Tools** > **UserSig Generation & Verification** on the left sidebar. Create two `UserID` values and their `UserSig` values, and copy the `UserID`, `Key`, and `UserSig` for subsequent logins.

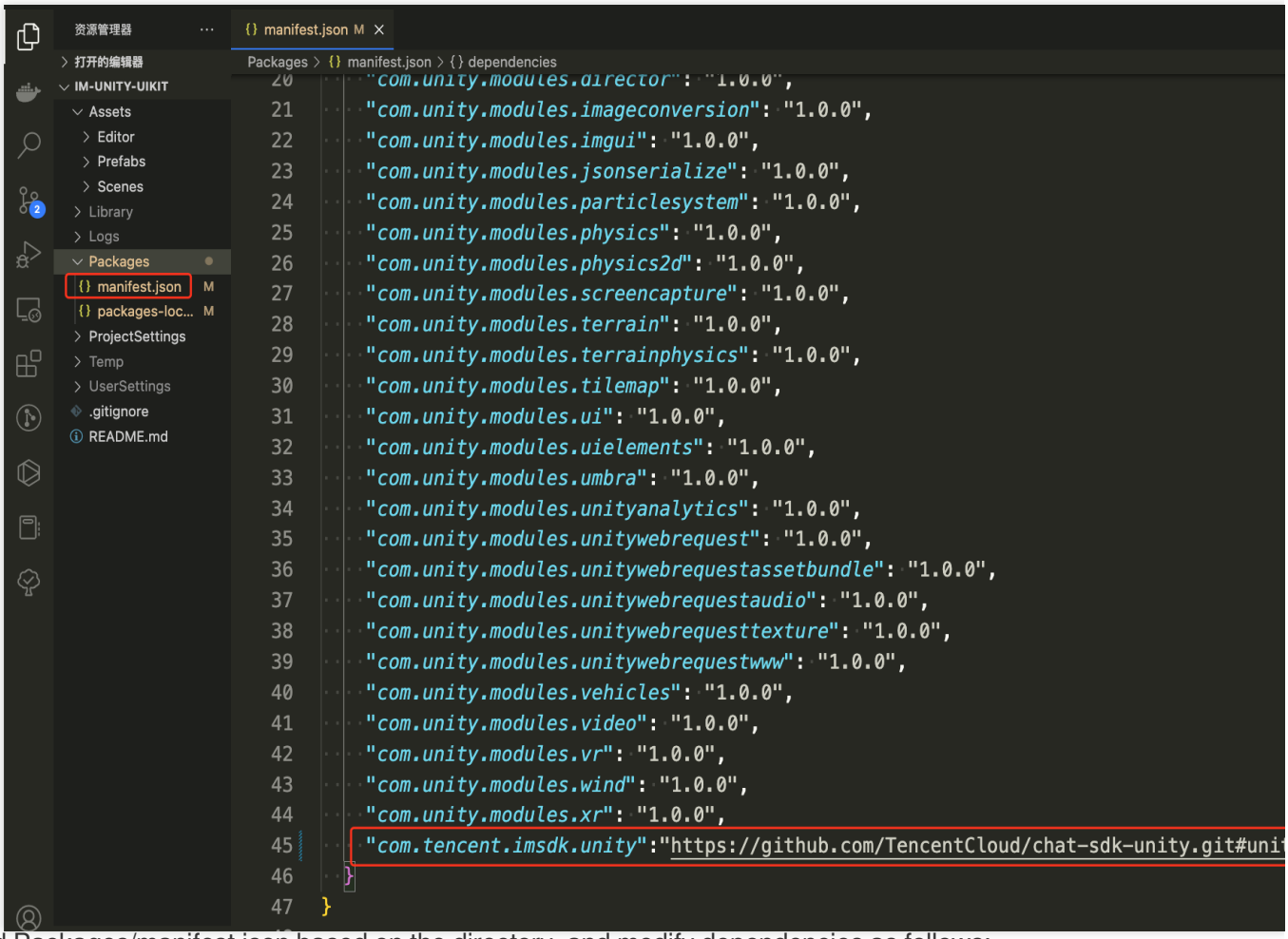
Note

The test account is for development and testing only. Before the application is launched, the correct method for generating a UserSig is to integrate the UserSig calculation code into your server and provide an application-oriented API. When UserSig is needed, your application can send a request to the business server for a dynamic UserSig. For more information, see Generating UserSig.



Part 2. Integrating Chat SDK into Your Unity Project

1. Use Unity to create a project and record the project directory, or open an existing Unity project.
2. Open the project with an IDE (such as Visual Studio Code):



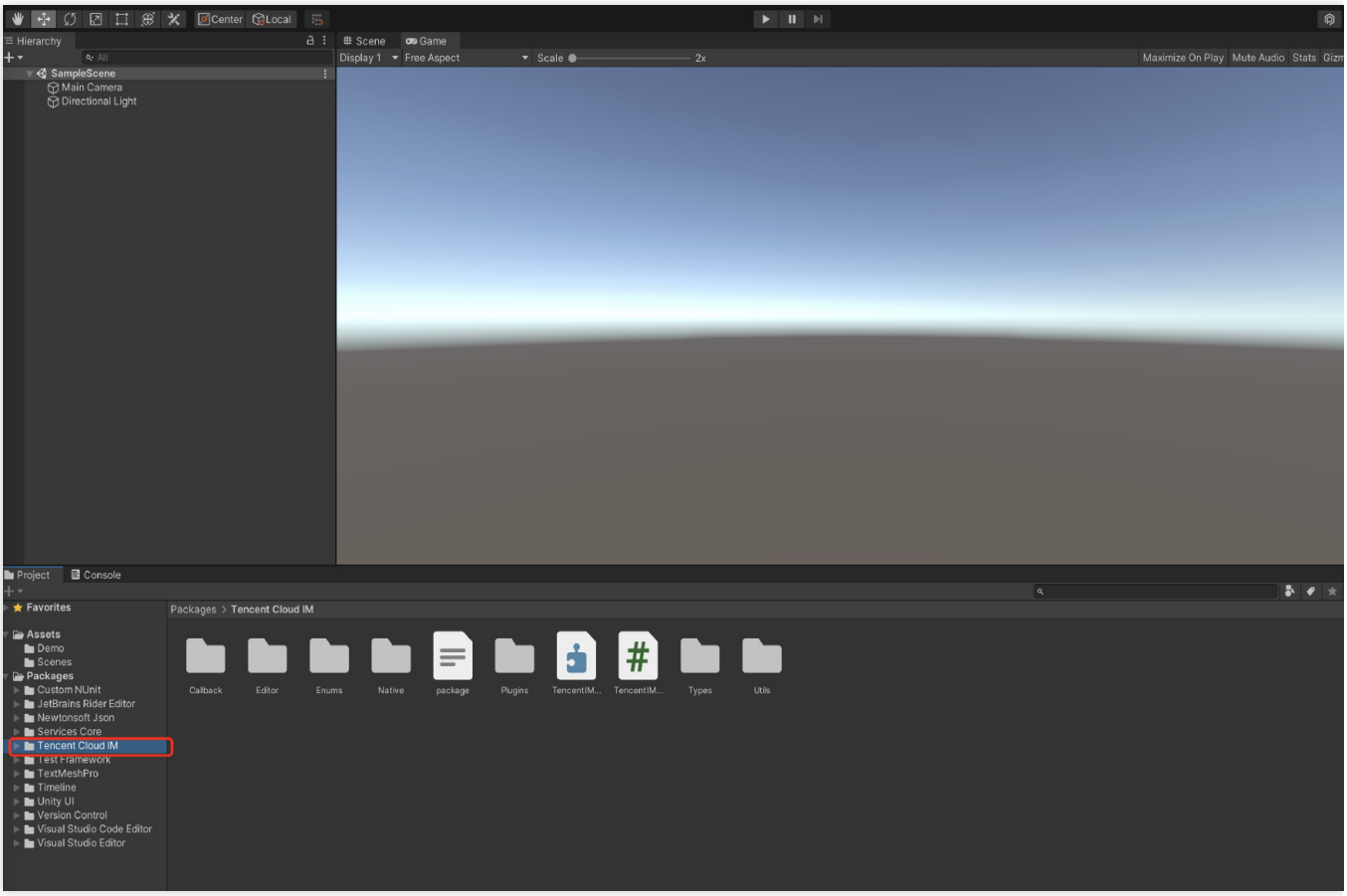
3. Find Packages/manifest.json based on the directory, and modify dependencies as follows:

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

To help you better understand Chat SDK APIs, we provide API examples to demonstrate how to call APIs and trigger listeners.

Part 3. Loading Dependencies

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



Part 4. Implementing Your Own UI

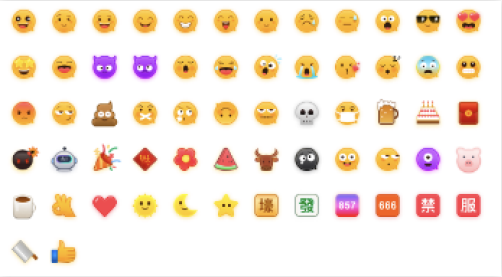
Prerequisites

You have created a Unity project or have a project that can be based on Unity, and have loaded Tencent Cloud Chat SDK.

Initializing the SDK

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.



Detailed documentation

Call `TencentIMSDK.Init` to initialize the SDK.

Pass in your `SDKAppID` .

```
using Com.Tencent.IM.Unity.UIKit;
using com.tencent.imsdk.unity;
using com.tencent.imsdk.unity.types;
using com.tencent.imsdk.unity.enums;
namespace Com.Tencent.IM.Unity.UIKit{
    public static void Init() {
      string SDKAppID = ""; // Get the `SDKAppID` from the Chat console
      SdkConfig sdkConfig = new SdkConfig();

      sdkConfig.sdk_config_config_file_path = Application.persistentDataPath + "/TI

      sdkConfig.sdk_config_log_file_path = Application.persistentDataPath + "/TIM-L

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

After `Init` , you can mount some listeners to the Chat SDK, mainly including those for network status and user information change. For more information, see here.

Logging in with a test account

Detailed documentation

Log in with one of the test accounts you created earlier.

Call the `TencentIMSDK.Login` method to log in with the test account.

If the returned `res.code` is `0` , the login is successful.

```
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
    });
  }
```

Note

The test account is for development and testing only. Before the application is launched, the correct method for generating a UserSig is to integrate the UserSig calculation code into your server and provide an application-oriented API. When UserSig is needed, your application can send a request to the business server for a dynamic UserSig. For more information, see Generating UserSig.

Sending a message

Detailed documentation

The following shows how to send a text message:

Sample code:

```
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); // messageId for getting

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
          // The message ID returned when the message is sent
}
```

Note

If sending fails, it may be that your `sdkAppID` doesn't support sending messages to strangers. In this case, you can disable the relationship check feature in the console.

Disable the friend relationship chain check here.

Obtaining the conversation list

Detailed documentation

Log in with the second test account to pull the conversation list.

The conversation list can be obtained in two ways:

1. Listen for the persistent connection callback to get message changes and update and render the historical message list in real time.
2. Call an API to get the message history at certain time points.

Common use cases include:

Getting the conversation list when the application starts and listening for the persistent connection callback to update the conversation list in real time.

Requesting the conversation list at certain time points

```
TIMResult res = TencentIMSDK.ConvGetConvList((int code, string desc, List<ConvInfo>
 // Process the async logic
});
```

At this point, you can see the message sent by the first test account in the previous step.

Listening for the persistent connection to get the conversation list in real time

Mount the conversation list listener, process the callback event, and update the UI.

Mount the listener.

```
TencentIMSDK.SetConvEventCallback((TIMConvEvent conv_event, List<ConvInfo> conv_lis
 // Process the callback logic
});
```

Process the callback event and display the latest conversation list on the UI.

Receiving messages

Detailed documentation

Messages can be received with the Chat SDK in two ways:

Listen for the persistent connection callback to get message changes and update and render the historical message list in real time.

Call an API to get the message history at certain time points.

Common use cases include:

After a new conversation is opened on the UI, request and display a certain number of historical messages.

Listen for the persistent connection to receive messages in real time and add them to the historical message list.

Requesting the historical message list at a time

To avoid affecting the pull speed, we recommend you limit the number of messages pulled to 20 per page.

You need to dynamically record the current number of pages for the next request.

Sample code:

```
// Pull historical one-to-one messages
// Do not set `msg_getmsglist_param_last_msg` for the first pull. It will pull the
// `msg_getmsglist_param_last_msg` can be the last message in the returned message
Message LastMessage = null;
string LastMessageID = "";
var get_message_list_param = new MsgGetMsgListParam();
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
  // handle callback logic
  List<Message> messages = Utils.FromJson<List<Message>>((string)parameters[1]);
   if (messages.Count > 0){
    LastMessage = messages[messages.Count - 1];
    LastMessageID.text = messages[messages.Count - 1].message_msg_id;
  }else {
    LastMessage = null;
    LastMessageID.text = "";
   }

});
// `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
  // handle callback logic
});
```

Listening for the persistent connection callback to get new messages in real time

After the historical message list is initialized, new messages are from the persistent connection `TencentIMSDK.AddRecvNewMsgCallback`.

After the `AddRecvNewMsgCallback` callback is triggered, you can add new messages to the historical message list as needed.

Sample code for binding a listener:

```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> message, string user_data) => {
  // Process new messages
});
```

At this point, you have completed the Chat module development, and now users can send and receive messages and enter different conversations.

You can develop more features, such as group, user profile, relationship chain, and local search.

For more information, see Integration Solution (No UI).

Part 5. Enabling Unity Support for WebGL

Tencent Cloud Chat SDK for Unity 1.8.1 or later supports building WebGL.
To enable support for web, you need to perform the following extra steps in addition to those for enabling support for Android and iOS:

Importing JS

Download the JS files below from Npm (you need to install nodejs, please refer to the nodejs official website) and place them in the folder where the project builds WebGL products.
index.js
modules/group-moudle.js
modules/relationship-module.js
modules/signaling-module.js
Open `index.html` and import the JS files as follows:

```
<script src="./index.js"></script>
<script src="./modules/group-module.js"></script>
<script src="./modules/relationship-module.js"></script>
<script src="./modules/signaling-module.js"></script>
```

FAQs

What platforms are supported?

iOS, Android, Windows, macOS, and WebGL are supported.

What should I do if clicking Build And Run for an Android device triggers an error, stating no available device is found?

Check that the device is not occupied by other resources. Alternatively, click **Build** to generate an APK package, drag it to the simulator, and run it.

What should I do if an error occurs during the first run for an iOS device?

If an error is reported for an iOS device after the demo configured as above is run, choose **Product** > **Clean** to clean the product, and build the demo again. You can also close Xcode and open it again, and then build the demo again.

What should I do if Unity v2019.04 reports the following error for iOS?

Library/PackageCache/com.unity.collab-proxy@1.3.9/Editor/UserInterface/Bootstrap.cs(23,20): error CS0117: 'Collab' does not contain a definition for 'ShowChangesWindow'
Choose **Window** > **Package Manager** on the toolbar of the Editor and downgrade Unity Collaborate to 1.2.16.

What should I do if Unity v2019.04 reports the following error for iOS?

Library/PackageCache/com.unity.textmeshpro@3.0.1/Scripts/Editor/TMP_PackageUtilities.cs(453,84): error CS0103: The name 'VersionControlSettings' does not exist in the current context
Open the source code and delete the code snippet of `|| VersionControlSettings.mode != "Visible Meta Files"` .

Is this a C# interface? How to use it without Unity?

Unity SDK is an SDK using C#, but because Unity SDK contains Unity features, it cannot be used directly in a C# environment.
If you need to use it in a C# environment, we provide a separate C# SDK nuget package. The usage method is the same as Unity SDK, you can refer to Unity SDK documentation for use.
Among them, the C# SDK only supports the PC side, and the unity SDK supports the mobile side.

Is there a UI that can be used directly?

The corresponding UIKit of untiy SDK and C# SDK is currently not provided.

How do I query error codes?

For Chat SDK API error codes, see Error Codes.

UE

Last updated：2024-11-04 17:00:02

This document describes how to quickly run the Chat demo for Unreal Engine.

Note:

Currently, the demo can be run on Windows, macOS, iOS, and Android.

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 free space after the IDE and required tools are installed. Install Visual Studio 2019.

Prerequisites

You have signed up for a Tencent Cloud account and completed identity verification.

Directions

Step 1. Create an app

1. Log in to the Chat console.

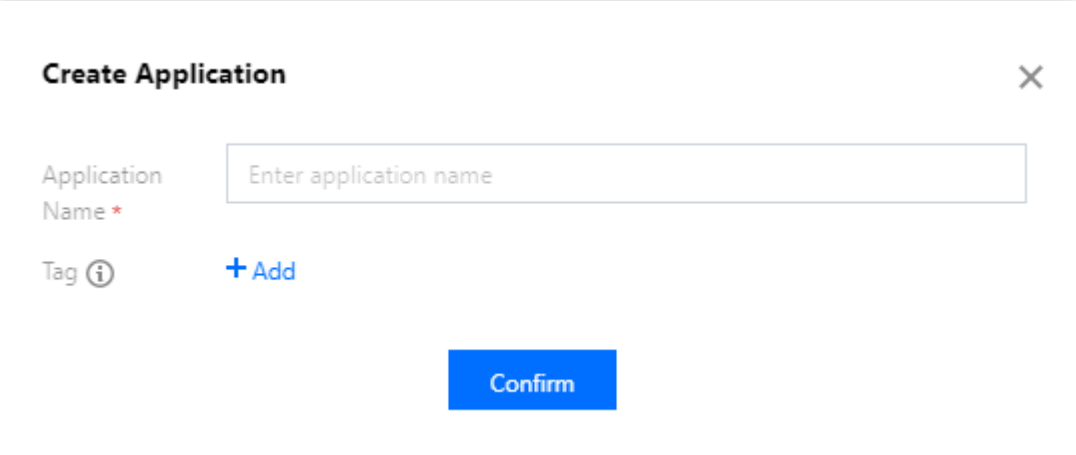
Note:

If you already have an app, record its SDKAppID and obtain key information.

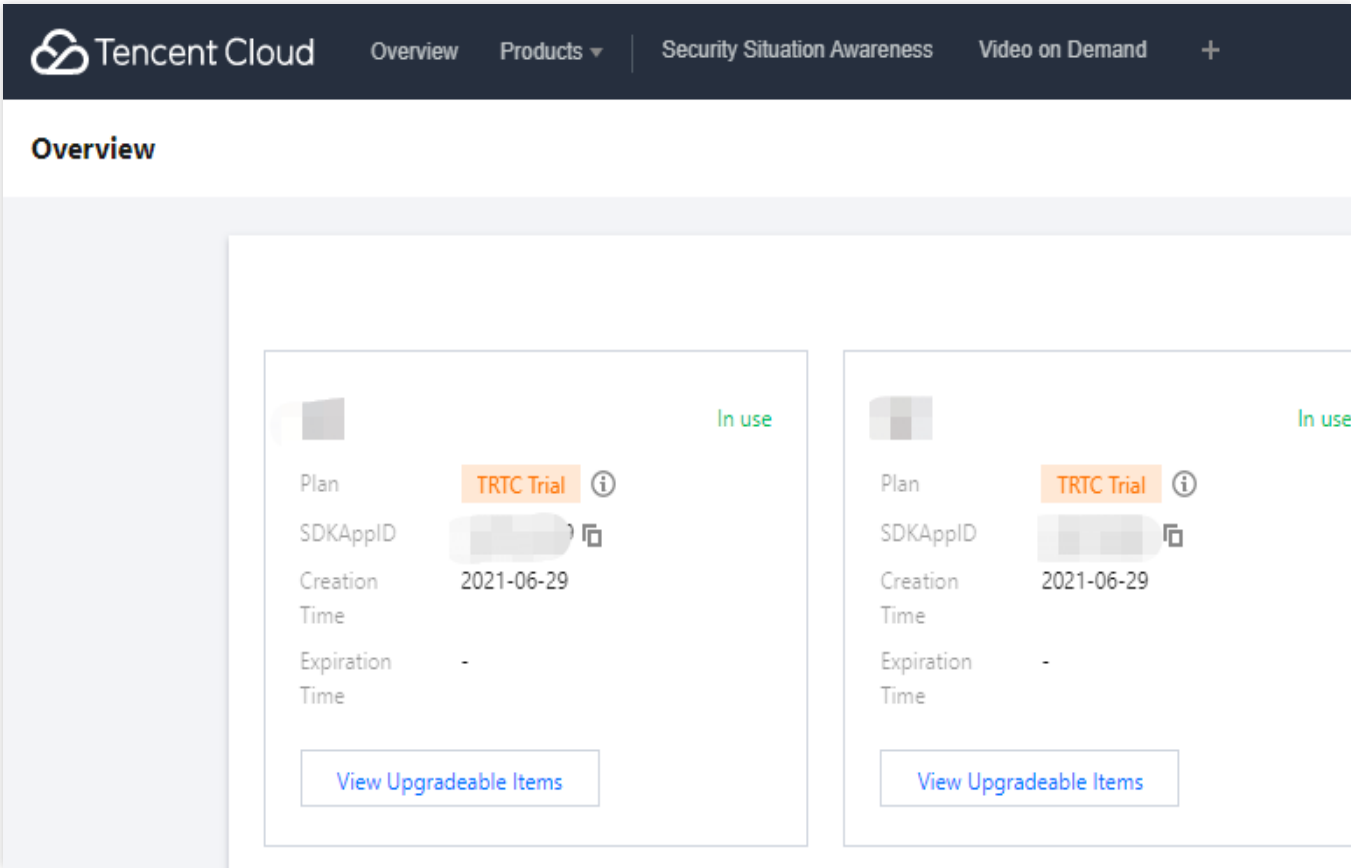
A Tencent Cloud account can create a maximum of 300 Chat apps, after which you need to disable and delete an unwanted app before creating a new one. **Once an app (along with its SDKAppID) is deleted, the service it**

provides and all its data are lost. Proceed with caution.

2. Click **Create Application**, enter your app name, and click **Confirm**.

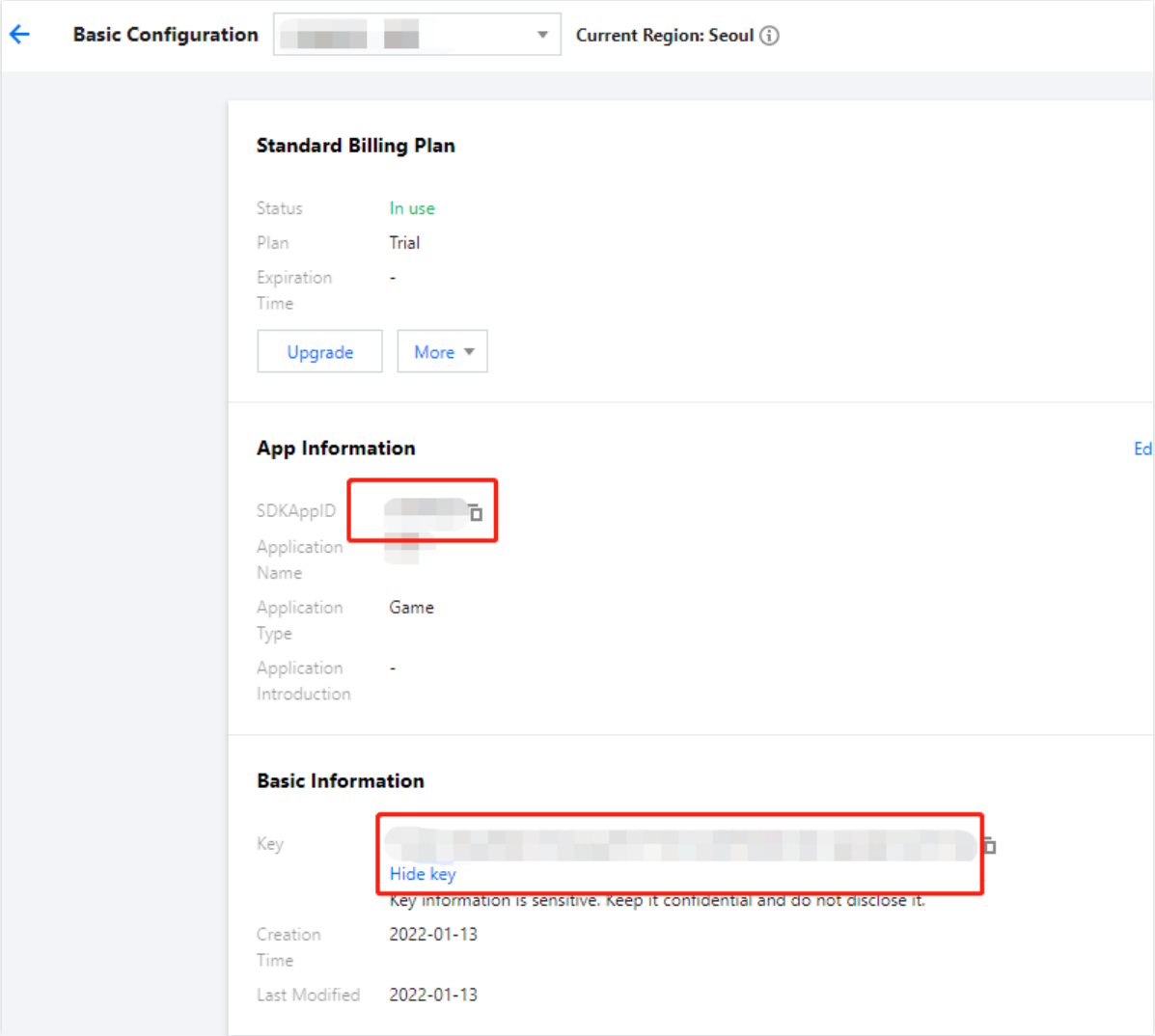


3. After creation, you can see the status, service version, SDKAppID, creation time, tag, and expiry time of the new app on the overview page of the console. Record the SDKAppID.



Step 2. Obtain key information

1. Click the app to go to its basic configuration page.



2. In the **Basic Information** area, click **Display key**, and then copy and save the key information.

Note:

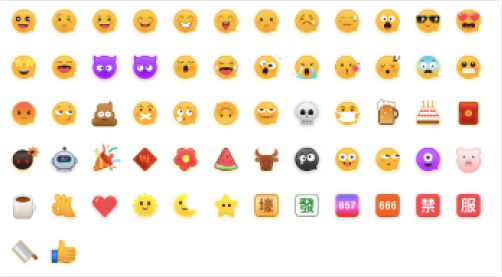
Make sure you keep your key information properly secure to prevent disclosure.

Step 3. Configure the demo project file

1. Download the Chat demo project from Download the Demo.

Note：

To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.



2. Find and open `/IM_Demo/Source/debug/include/DebugDefs.h` .
3. Set parameters in `DebugDefs.h` as follows:

Note:

In this document, the method to obtain UserSig is to configure a SECRETKEY in the client code. In this method, the SECRETKEY is vulnerable to decompilation and reverse engineering. Once your SECRETKEY is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see Generating UserSig on the Server.

Step 4. Compile, package, and run the project

1. Double-click to open `/IM_Demo/IM_Demo.uproject` .
2. Compile, run, and test the project.

macOS

Windows

iOS

Android

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

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

Package Project

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

1. For development and testing, see Android Quick Start.
2. For packaging, see Packaging Android Projects.

Chat API Documentation for Unreal Engine

For more information on APIs, see API Overview.

FAQs

What should I do if the error "Attempt to construct staged filesystem reference from absolute path" occurs on Android?

Close the project in UE4, open CMD, and run the following commands:

```
adb shell

cd sdcard

ls (you should see the UE4Game directory listed)

rm -r UE4Game
```

Compile your project again.

Flutter

Last updated：2025-03-18 10:08:34

This document describes how to integrate the SDK for Flutter.

Try Out the Demo

Before getting started, you can try out the Flutter Chat Demo with the embedded UIKit here.

Mobile App Android & iOS	**Web** The QR Code & 'Visit' button direct to the same page		**Windows**	**macOS**
			Windows 10+	

Environment Requirements

Environment	Version
Flutter	Flutter 3.0.0 or later for the Chat SDK; Flutter 3.24.0 or later for the TUIKit component library.
Android	Android Studio Dolphin | 2021.3.1 or later; and devices with Android 7.0 or later for apps.
iOS	Xcode 12.0 or later. Ensure that your project has a valid developer signature.

Supported Platforms

We offer a set of Chat SDK and UIKit for all Flutter platforms, allowing you to run one set of code on all platforms.

Platform	Low-level SDK (tencent_cloud_chat_sdk)	UIKit (tencent_cloud_chat_common, tencent_cloud_chat_message,

		tencent_cloud_chat_conversation, tencent_cloud_chat_contact, tencent_cloud_chat_sticker, tencent_cloud_chat_message_reaction, tencent_cloud_chat_text_translate, tencent_cloud_chat_sound_to_text, tencent_cloud_chat_push, tencent_calls_uikit)
iOS	Supported	Supported
Android	Supported	Supported
HarmonyOS NEXT	Supported in version 8.4.6675-beta.2	Developing
macOS	Supported from v4.1.9	Supported
Windows	Supported from v4.1.9	Supported
Web	Supported from v4.1.1+2	Supported

Directions

1. Create an App

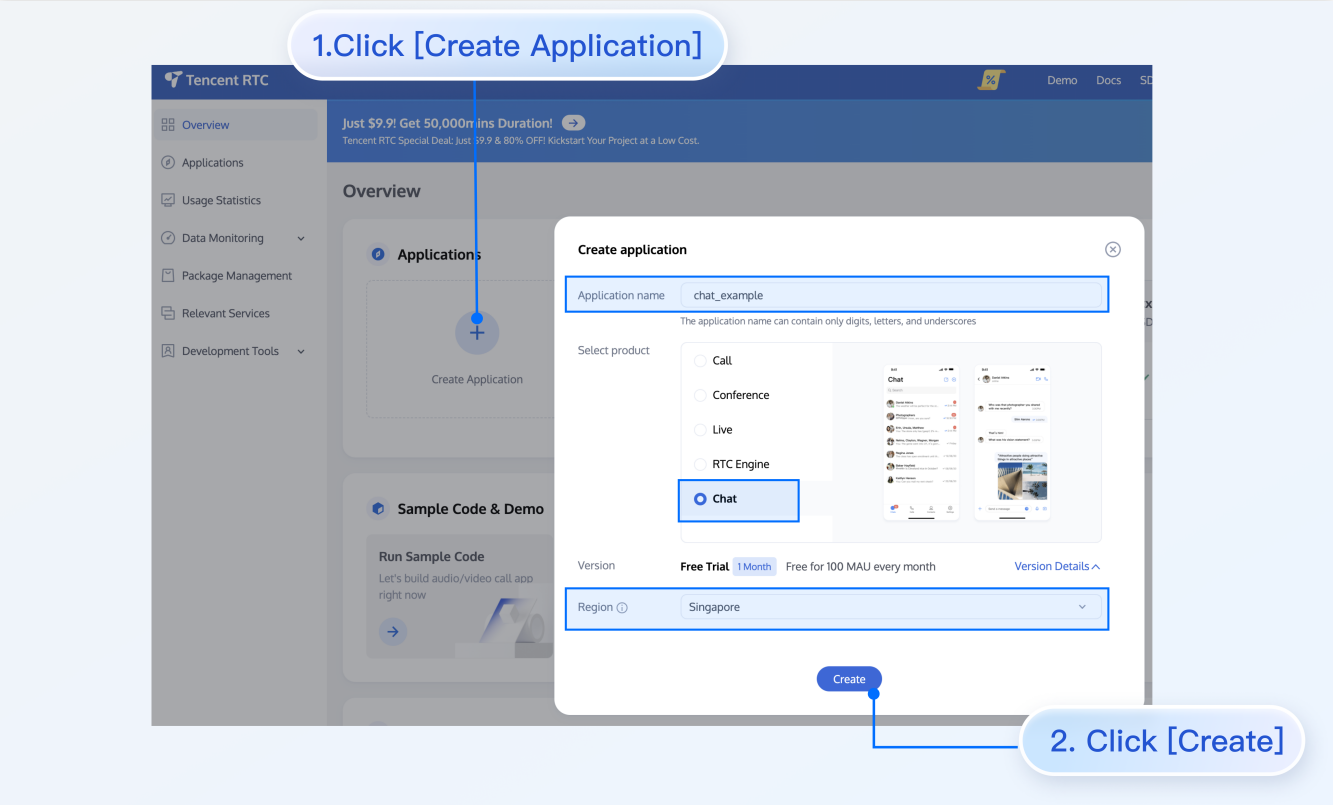
1. Log in to the Chat Console. If you already have an app, record its SDKAppID and SDKSecretKey.

Note：

1. A Chat account can create a maximum of 300 Chat apps. If you want to create a new app, you can disable and delete an unwanted app first.

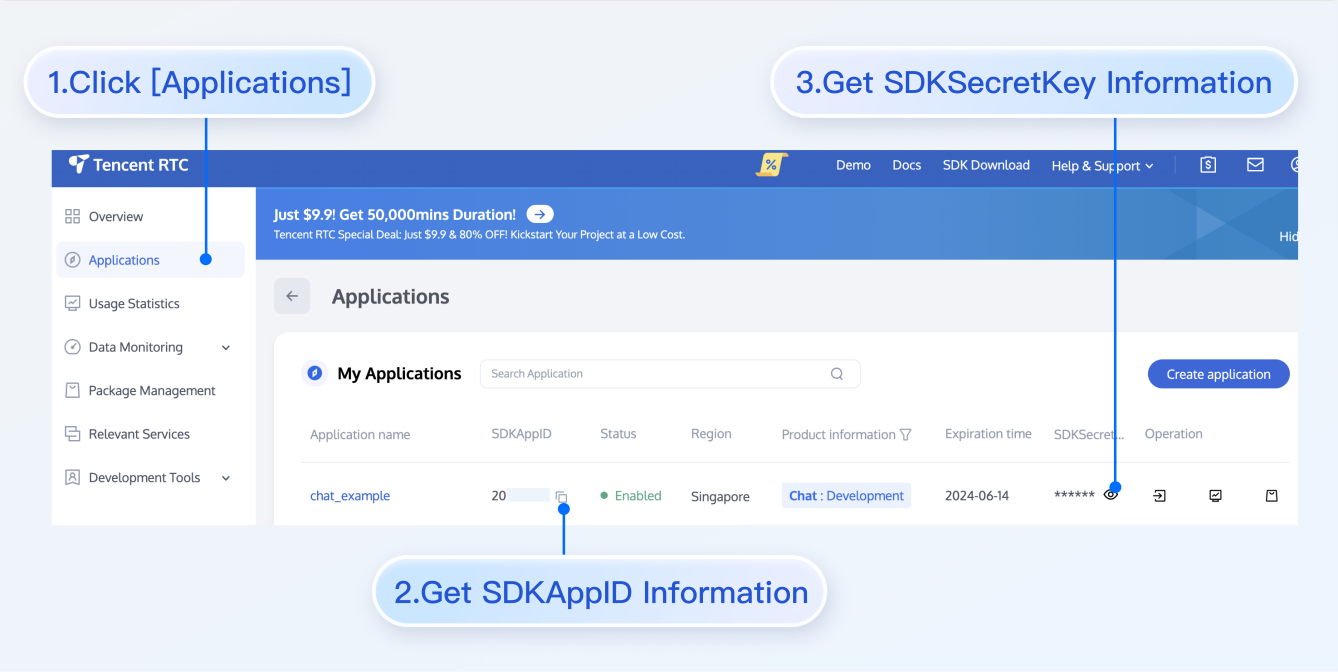
2. **Once an app (along with its SDKAppID) is deleted, the service it provides and all its data are lost. Please operate with caution.**

2. Click `Create Application` , enter your Application name, product, Region, and click `Create` .



2. Obtain SDKAppID, SDKSecretKey and Login User

After creation, you can view the newly created app's Status, SDKAppID, Expiration time, etc., on the Applications page:



Record the SDKAppID and SDKSecretKey from the Application Information.

Danger:

Keep the SDKSecretKey properly to prevent disclosure.

3. Download and Configure the Demo

1. Download the source code and install dependencies:

```
# Clone the code
git clone https://github.com/TencentCloud/chat-demo-flutter.git

# Checkout the 'v2' branch
git checkout v2

# Clean the project. Important
flutter clean

# Install dependencies
flutter pub get
```

2. Configure the user info for login.

Open `lib/config.dart` , and specify the `sdkappid` and `key` obtained in the previous step.

sdkAppID：set it to the SDKAppID obtained above.

key：set it to the SDKSecretKey obtained above.

Warning：

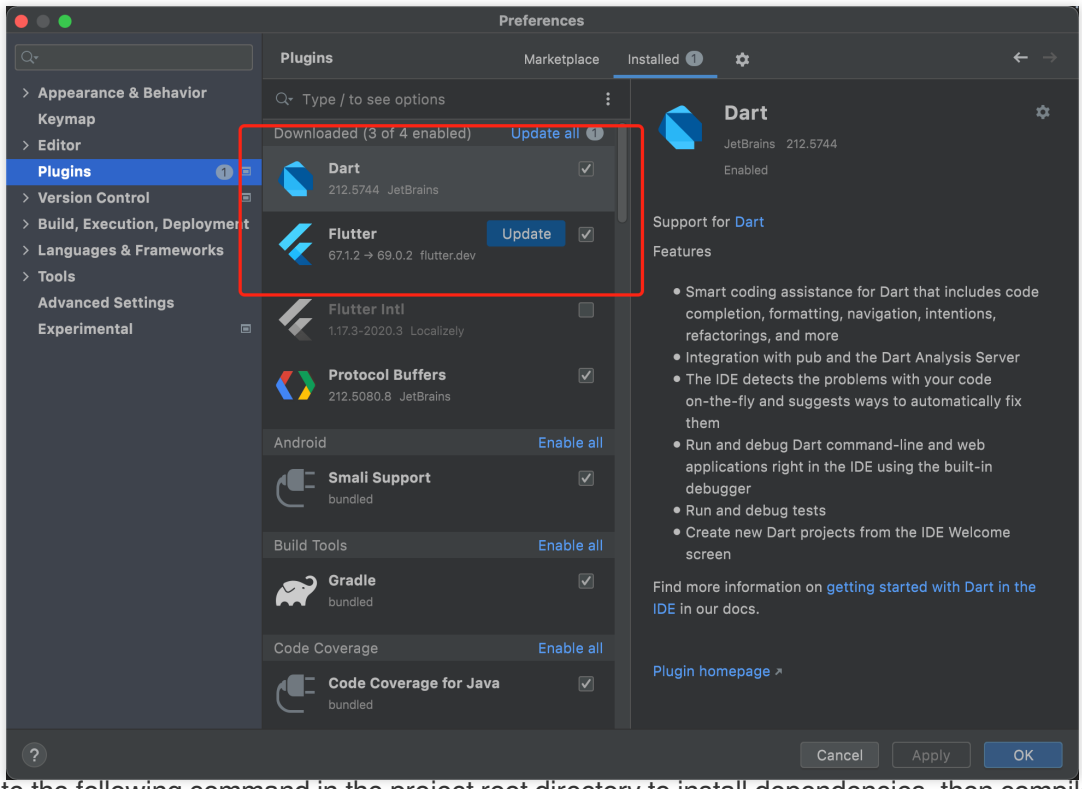
In this document, the method to obtain UserSig is to configure a SECRETKEY in the client code. In this method, the SECRETKEY is vulnerable to decompilation and reverse engineering. Once your SECRETKEY is disclosed, attackers

can steal your Tencent Cloud traffic. Therefore, this method is only suitable for locally running a demo project and feature debugging.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an app-oriented API. When `UserSig` is needed, your app can send a request to the business server to obtain a dynamic `UserSig`. For more information, see How to Generate UserSig on the Server.

4. Compile and Run the Demo

Import the demo project with Android Studio, and install the Flutter and Dart plugins.



Execute the following command in the project root directory to install dependencies, then compile and run.

```
flutter pub get
```

Platform Configuration

HarmonyOS NEXT

Starting from version 8.4.6675-beta.2, the No-UI SDK (tencent_cloud_chat_sdk) supports HarmonyOS NEXT. This adaptation is developed based on the Flutter 3.22 version optimized for HarmonyOS.

HarmonyOS has adapted many third-party Flutter libraries. For the tencent_cloud_chat_sdk, the only required third-party library is path_provider. Therefore, you need to override the dependency in your **project's root** pubspec.yaml file to use the HarmonyOS-adapted version of path_provider.

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

Web

To enable support for web, you need to perform the following extra steps in addition to those for enabling 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.

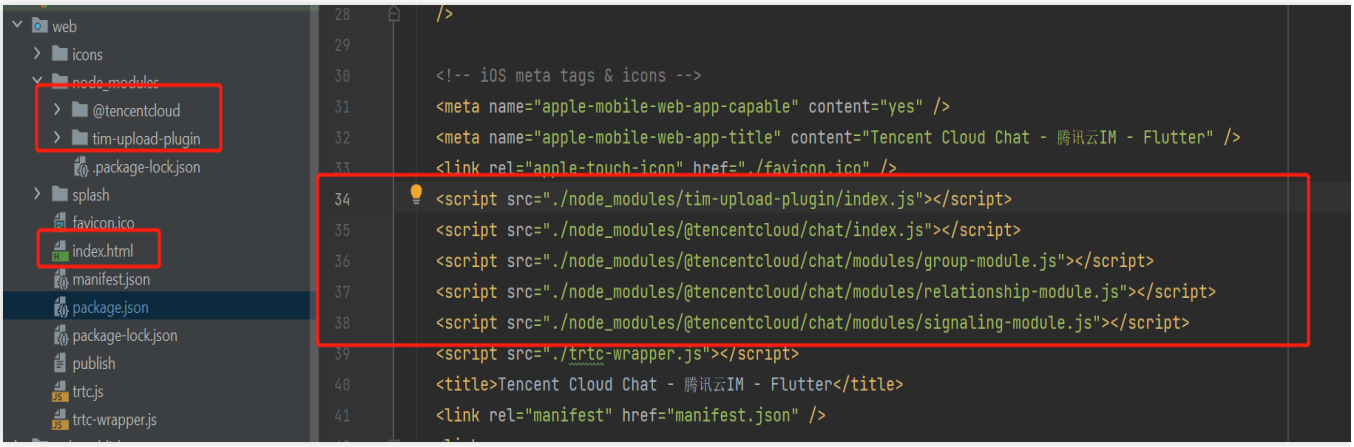
Importing JS

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 run `npm` or `yarn` to install relevant JS dependencies. Initialize the project as instructed.

```
cd web

npm init

npm i @tencentcloud/chat

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/@tencentcloud/chat/index.js"></script>
<script src="./node_modules/@tencentcloud/chat/modules/group-module.js"></script>
<script src="./node_modules/@tencentcloud/chat/modules/relationship-module.js"></sc
<script src="./node_modules/@tencentcloud/chat/modules/signaling-module.js"></scrip
```

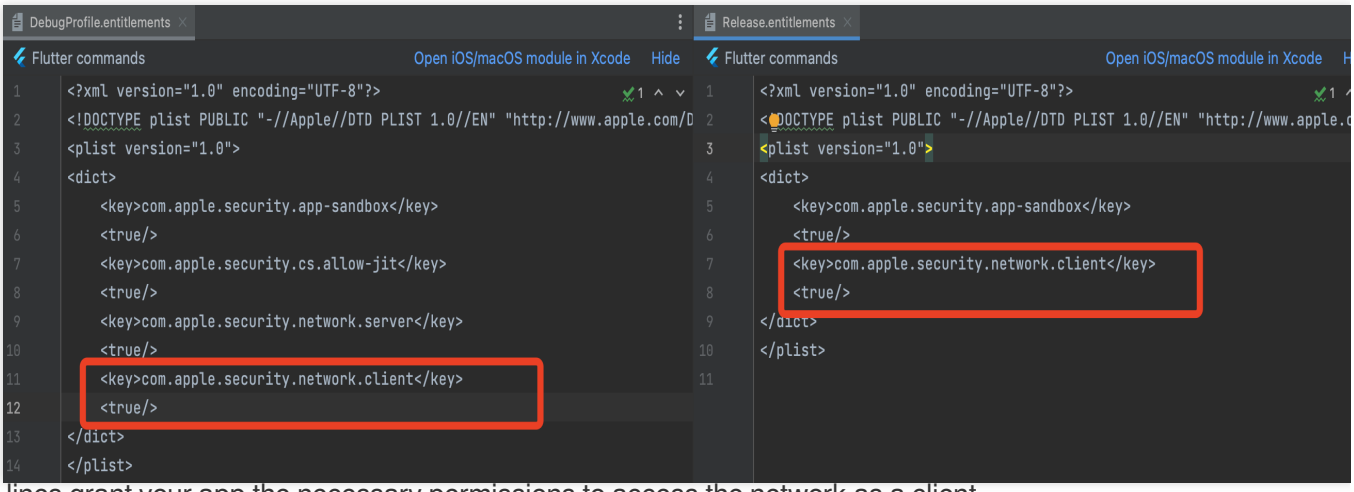


macOS

Additional configurations are required for the macOS platform. Follow the steps below to configure the macOS platform:

1. Open the `macos/Runner/DebugProfile.entitlements` and `macos/Runner/Release.entitlements` files in your project.
2. Add the following lines to each file:

```
<key>com.apple.security.network.client</key>
<true/>
```



These lines grant your app the necessary permissions to access the network as a client.

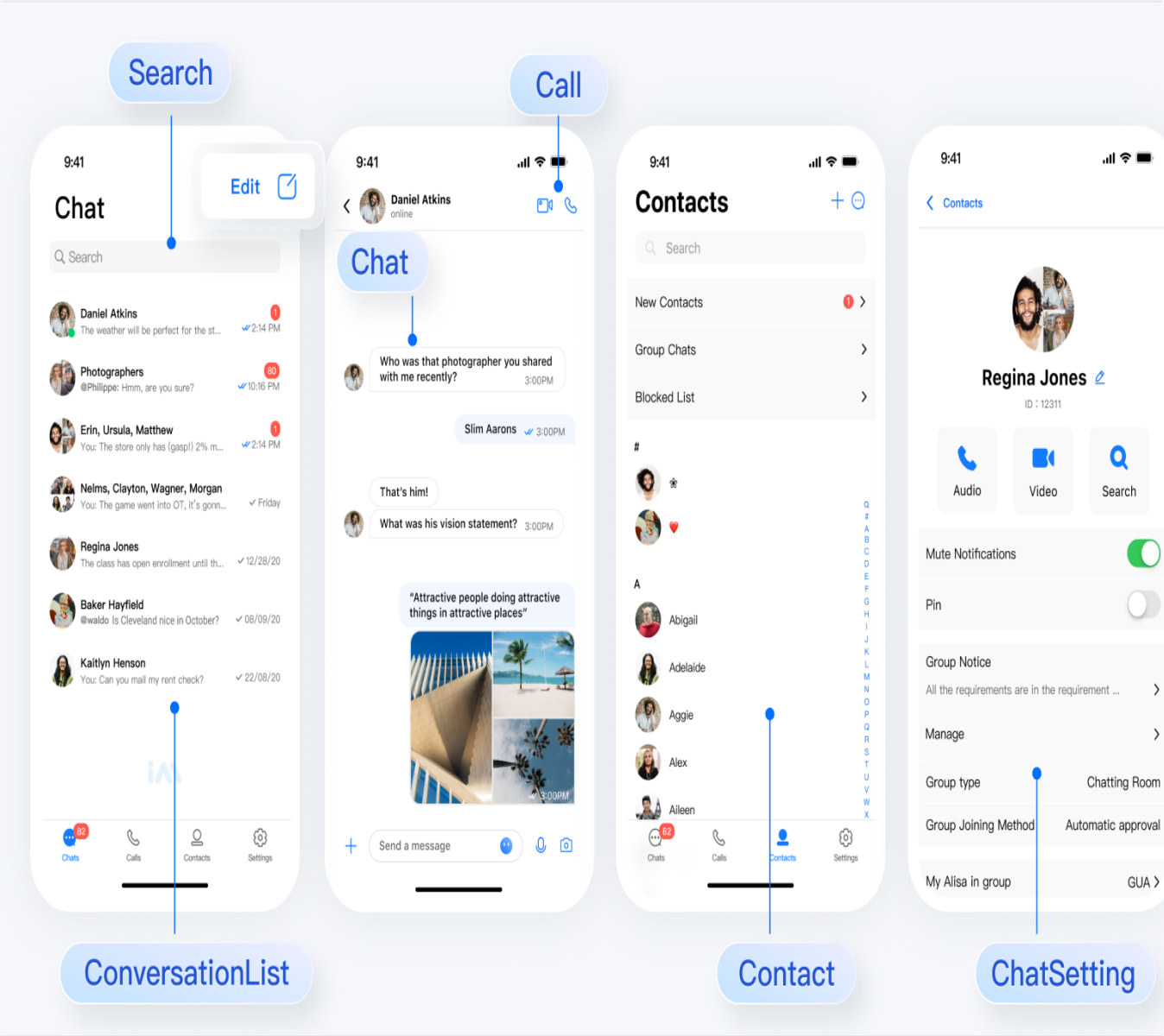
This configuration is essential for ensuring proper communication between your app and the backend services on the macOS platform.

React Native

Last updated：2024-12-13 17:42:55

Introduction to chat-uikit-react-native

chat-uikit-react-native is a React Native UI component library based on Tencent Cloud Chat SDK. It provides universal UI components, including conversation, chat, and group features. With these well-designed UI components, you can quickly build elegant, reliable, and scalable chat applications. The UIKit interface developed with React Native is more in line with the usage habits of overseas customers and supports internationalization. If your business needs to expand overseas, you are welcome to integrate it. For details, refer to the open source code.
The interface effect of chat-uikit-react-native is shown below:



Environment Requirements

React Native 0.75.0

Node.js version 18+

Xcode version 14.0 or above

Android Studio

Configuring the development environment

If you are developing a React Native project for the first time, please refer to the steps on the React Native official website set-up-your-environment to configure your development environment.
If you encounter any environment issues during project creation or compilation, you can run `npx react-native doctor` for an environmental diagnosis.

Running a Demo

Step 1: Download the Source Code

```
git clone https://github.com/TencentCloud/chat-demo-react-native
```

```
cd chat-demo-react-native/Demo
```

Install Using **`yarn`** **(Recommended)**

```
yarn install
```

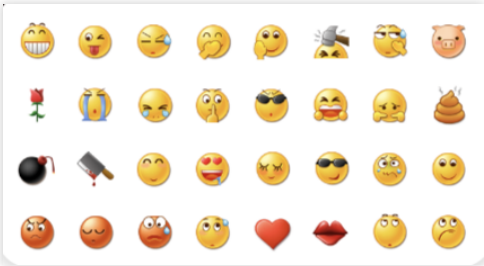
Or Install Using **`npm`**

```
npm i --legacy-peer-deps
```

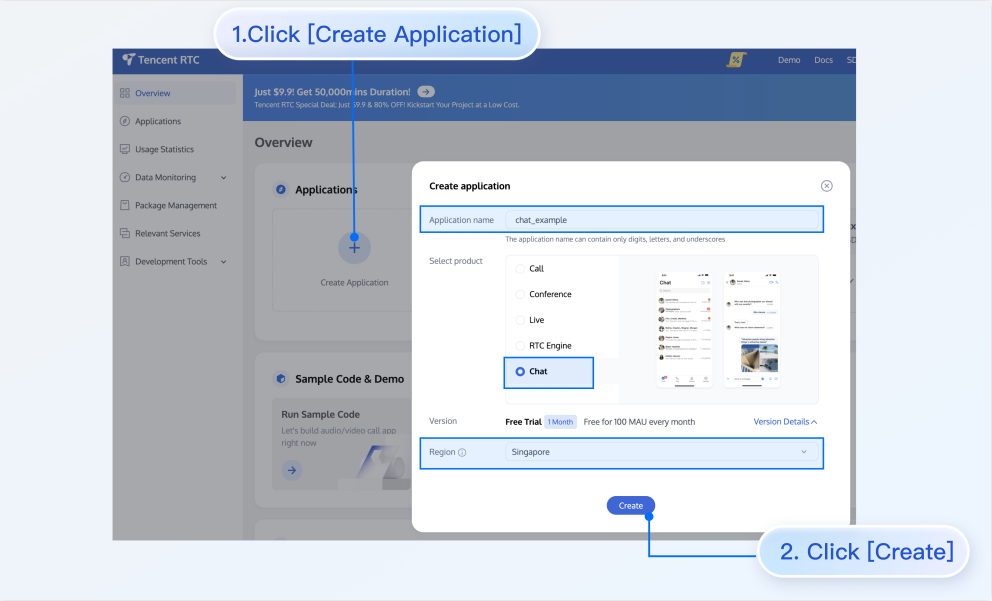
Step 2: Configure the Demo

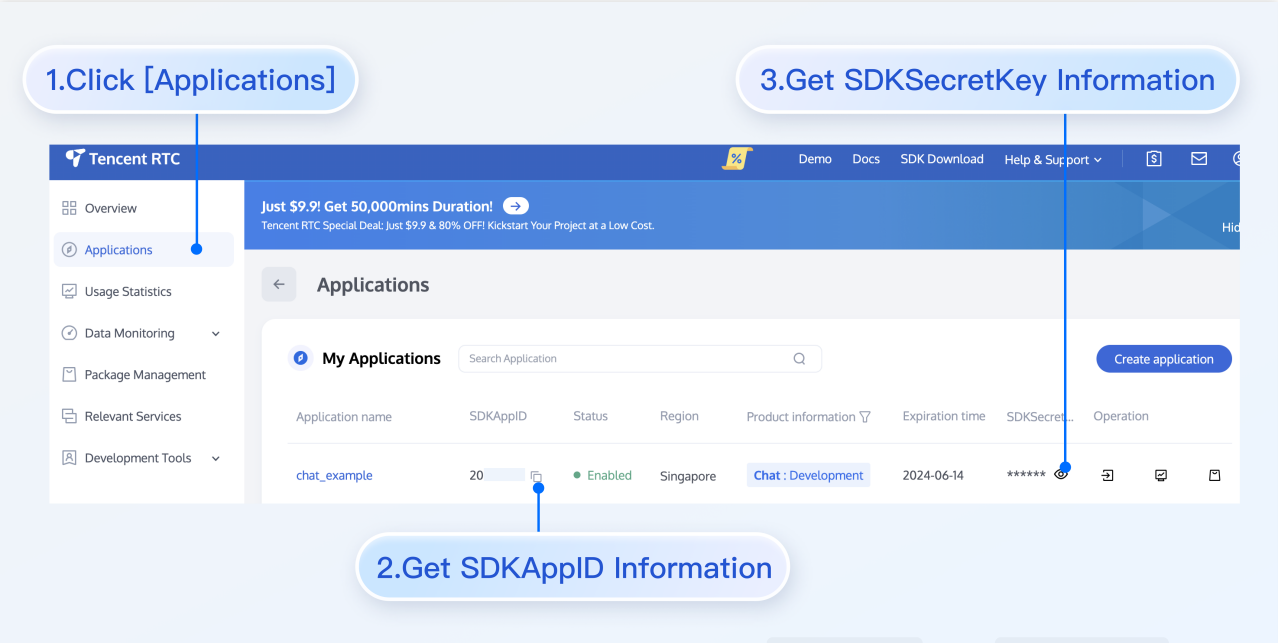
Note:

To respect the copyright of emoji designs, the IM Demo/TUIKit project does not include large emoji elements. Please replace them with your own designed or copyrighted emoji packs before official commercial launch. The **default QQ emojis** shown below are copyrighted by Tencent and are not available for licensing. Thank you for your understanding and awareness.



1. Open the Demo project, the GenerateTestUserSig.js file under the ./debug directory.
2. Set `SDKAPPID` and `SECRETKEY` in the `GenerateTestUserSig.js` file, which can be obtained from the Chat Console. Click the target application card to enter its configuration page.





3. In the area **shown in the figure**, click **copy** to replace the original `SDKAPPID` and `SECRETKEY` in the `GenerateTestUserSig.js` file.

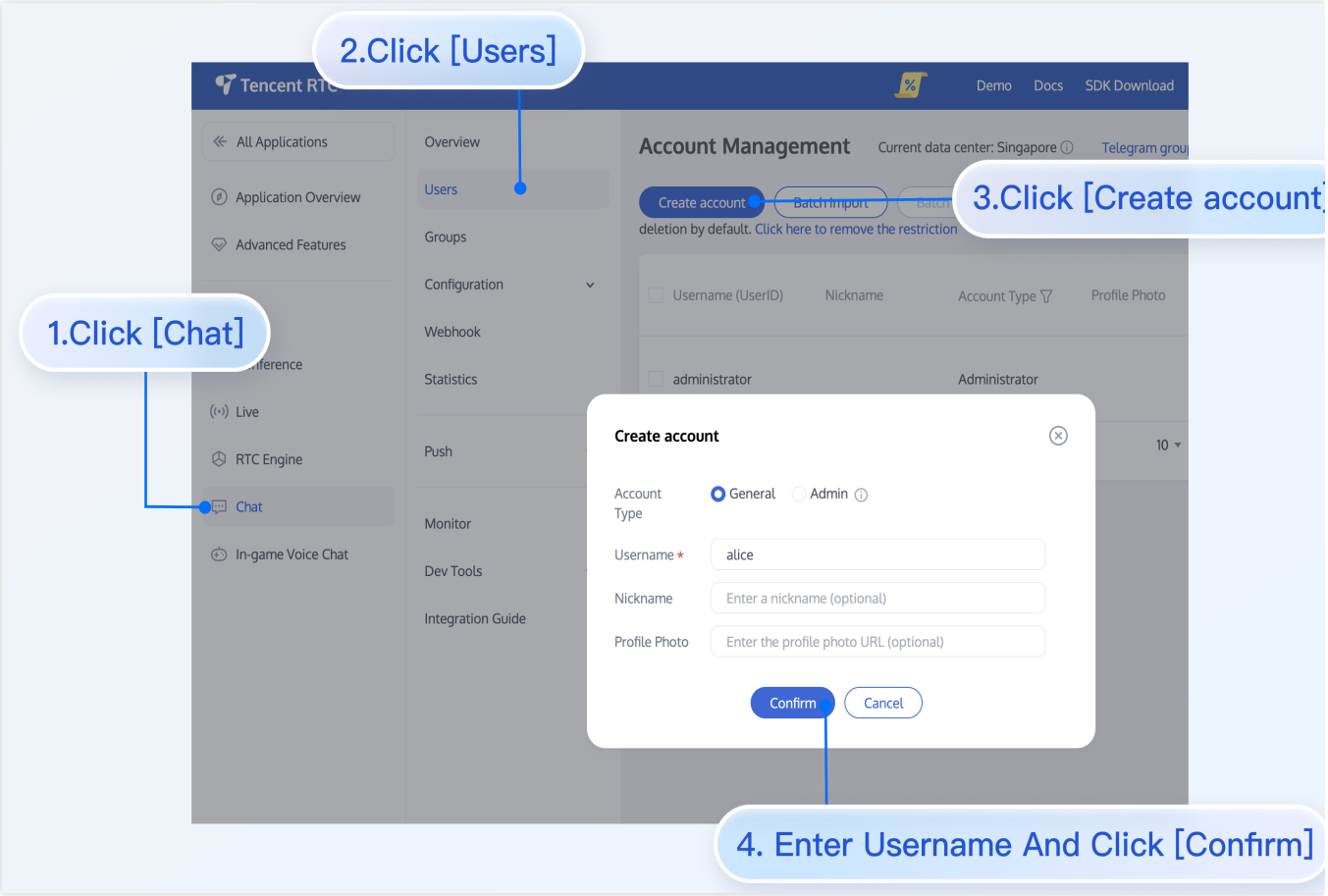
4. Enter the account management page of the application, create an account and obtain the userID, which will be used as the test user for sending messages later.

Click to enter the Application you created above, you will see the `Chat` product entry in the left sidebar, click to enter.

After entering the Chat product sub-page, click `Users` to enter the User Management page.

Click `Create account` to pop up the account creation information filling box. If it is just a regular member, we recommend selecting the `General` type.

For a better experience with message sending and receiving features, create userID (test_1, test_2).



Step 3: Launching the Project

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

Android

iOS

1. Enable Developer Mode on the phone and turn on the **USB Debugging** switch.
2. Connect the phone with a USB cable, it is recommended to choose the **Transfering File** option, **do not choose the Charge Only option**.
3. After confirming the phone is successfully connected, execute `npm run android` to compile and run the project.

```
npm run android
```

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

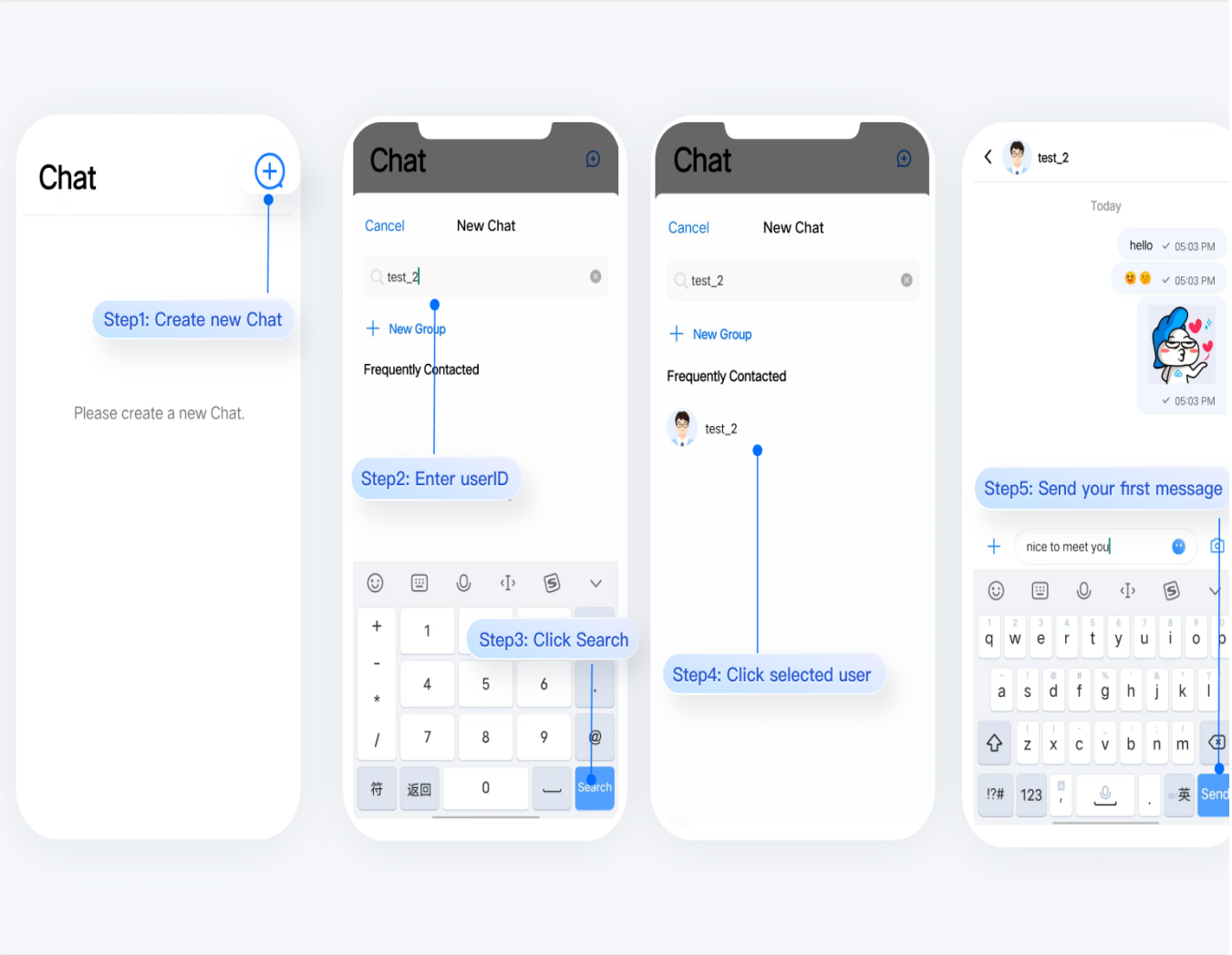
```
cd ios
pod install
```

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

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

Step 4: Sending Your First Message

1. After the project starts, click Initiate Session in the top left corner.
2. Enter the conversation initiation window. In the search bar, enter the userID created in Step 2 (**test_2**), select it, and open the conversation.
3. Enter the message in the input box and click send.



Exchange and Feedback

Join Telegram Technical Support Group or WhatsApp Communication Group for professional engineer support to solve your problems.

FAQs

1. If you encounter an error as shown in the figure when running npm run android, please reset the environment variables in the project root directory.

```
BUILD FAILED in 2s
error Failed to install the app. Command failed with exit code 1: ./gradlew app:installDebu
PreactNativeDevServerPort=8081 FAILURE: Build failed with an exception. * What went wrong:
Could not determine the dependencies of task ':app:compileDebugJavaWithJavac'.
> SDK location not found. Define a valid SDK location with an ANDROID_HOME environment vari
```

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

2. If you encounter node environment variable issues when executing the Build command in Xcode, please follow these steps:

```
[Warning] You need to configure your node path in the `".xcode.env" file` environment.  You
n set it up quickly by running: `echo export NODE_BINARY=$(command -v node) > .xcode.env`
 the ios folder. This is needed by React Native to work correctly.  We fallback to the DEPR
TED behavior of finding `node`. This will be REMOVED in a future version.  You can read mor
bout this here: https://reactnative.dev/docs/environment-setup#optional-configuring-your-er
onment
```

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

Reference Documentation

Related to UIKit:

chat-uikit-react-native npm

Quick Integration Documentation for UIKit

Referencing the ChatEngine API documentation for more features

chat-uikit-engine API Manual

chat-uikit-engine npm

Add Flutter to your existing app

Last updated：2025-06-10 10:59:48

Adding in-app chat and call modules to an Android/iOS app can be necessary as a business grows. For example, chat modules can be added to video, e-commerce, or entertainment apps to facilitate communication between users. However, adding these modules can be time-consuming and lead to an inconsistent user experience. One solution is to integrate Tencent Cloud Chat with Flutter, which allows for coding once and deploying to all platforms.
If it's not practical to rewrite the entire application in Flutter, Flutter can be integrated as a library or module. This module can then be imported into the existing Android or iOS app to render a part of the app's UI in Flutter.
This can greatly reduce your workload and allow you to quickly integrate chat and call features of the Chat SDK into your native apps.

Note：
To respect the copyright of emoji designs, the Chat Demo/TUIKit project does not include cutouts of large emoji elements. Please replace them with your own designs or other emoji packs for which you hold the copyright before officially launching for commercial use. **The default smiley face emoji pack shown below is copyrighted by Tencent RTC** and is available for licensed use for a fee. If you need to obtain a license, please contact us.

Environment requirements

Platform	Version
Flutter	Flutter 2.2.0 or later for the Chat SDK; Flutter 2.10.0 or later for the TUIKit integration component library.
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.
Tencent Cloud Chat SDK	tencent_im_sdk_plugin 5.0 or later, tim_ui_kit 0.2 or later.

Sample Code：

The source code of the sample app can be found at GitHub repo.

What you need to know first

Before starting, we recommend getting to know the Tencent Cloud Chat SDK for Flutter, TUIKit, and the principles of hybrid app development with Flutter.

Tencent Cloud Chat

Overall

Before starting, you should be familiar with Tencent Cloud Chat SDK for Flutter and how it is used.
Chat offers two SDKs. One is an SDK with no UI components included. The other is UIKit, which includes UI components.
To demonstrate hybrid app development, this tutorial is written for development using TUIKit.
To learn more about the Chat SDK, see Get Started.

Modules of Tencent Cloud Chat

Tencent Cloud Chat includes two main modules: Chat and Call.
The Chat module enables you to send and receive messages, management user relationships, etc.
The Call module enables you to send and receive audio and video calls, including one-to-one call and group calls.

Adding Flutter to Native Apps

The basic principle behind hybrid app development with Flutter is to embed a Flutter module into your existing native app as a `subproject`. Because of the cross-platform nature of Flutter, you only need to develop the Flutter module once, and it can be added to both Android and iOS projects.
To launch a Flutter screen from an existing iOS/Android, you start a FlutterEngine and a FlutterViewController/FlutterActivity.
The `FlutterEngine` serves as a host to the Dart VM and your Flutter runtime, and the `FlutterViewController` / `FlutterActivity` attaches to a FlutterEngine to pass input events into Flutter and to display frames rendered by the `FlutterEngine`.
The `FlutterEngine` may have the same lifespan as your `FlutterViewController` / `FlutterActivity` or outlive your `FlutterViewController` / `FlutterActivity`.
Method Channel can be used to communicate between the native app and the Flutter module, for example when passing current user information, transmitting audio and video call data, or triggering offline push notifications. To trigger the method of the other end, use `invokeMethod` and to listen for the method callback from the other end using the pre-mounted `MethodCallHandler`.

Adding a Flutter Module to an Android App

Detailed documentation

This method involves adding the Flutter module as a dependency of your existing app in Gradle. There are two ways to achieve this. The first is using the AAR mechanism to create generic Android AARs as intermediaries that package your Flutter module. This is good when your downstream app builders don't want to have the Flutter SDK installed. But, it adds one more build step if you build frequently.

The other is using the source code subproject mechanism, which is a convenient one-click build process, but requires the Flutter SDK. This is the mechanism used by the Android Studio IDE plugin.

Option A - Depend on the Android Archive (AAR)

This option packages your Flutter library as a generic local Maven repository composed of AARs and POMs artifacts. This option allows your team to build the host app without installing the Flutter SDK. You can then distribute the artifacts from a local or remote repository.

It's recommended to use this option for the release version of your app.

Steps:

Run the following command on your Flutter module.

```
flutter build aar
```

Then, follow the on-screen instructions to integrate.

Your app now includes the Flutter module as a dependency.

Option B - Depend on the module's source code

This option enables a one-step build for both your Android project and Flutter project.

This option is convenient when you work on both parts simultaneously and rapidly iterate, but your team must install the Flutter SDK to build the host app.

It's recommended to use this option when development and debugging.

Steps:

Include the Flutter module as a subproject in the host app's `settings.gradle` :

```
// Include the host app project.
include ':app'                                    // assumed existing content

// Add the following lines to your project
setBinding(new Binding([gradle: this]))
evaluate(new File(
settingsDir.parentFile,
'tencent_chat_module/.android/include_flutter.groovy'
```

```
))
```

Introduce an `implementation` dependency on the Flutter module from your app:

```
dependencies {
implementation project(':flutter')
}
```

Your app now includes the Flutter module as a dependency.

Adding a Flutter Module to an iOS App

Detailed documentation

There are two ways to embed Flutter in your existing application.

Use the CocoaPods dependency manager and install Flutter SDK. (Recommended)

Create frameworks for the Flutter engine, your compiled Dart code, and all Flutter plugins. Manually embed the frameworks, and update your existing application's build settings in Xcode.

Note:

Your app does not run on a simulator in Release mode because Flutter does not yet support outputting x86/x86_64 ahead-of-time (AOT) binaries for your Dart code. You can run in Debug mode on a simulator or a real device, and Release on a real device.

To run your app on a simulator follow the instructions in the bottom of section embed the frameworks.

Option A - Embed with CocoaPods and the Flutter SDK

This method requires every developer working on your project to have a locally installed version of the Flutter SDK.

Simply build your application in Xcode to automatically run the script to embed your Dart and plugin code.

This allows rapid iteration with the most up-to-date version of your Flutter module without running additional commands outside of Xcode.

It's recommended to use this option for development and debugging.

Steps:

Add the following lines to your `Podfile`:

```
// The path of your Flutter module
flutter_chat_application_path = '../tencent_chat_module'

load File.join(flutter_chat_application_path, '.ios', 'Flutter',
'podhelper.rb')
```

For each Podfile target that needs to embed Flutter, call `install_all_flutter_pods(flutter_application_path)`.

```
target 'MyApp' do
install_all_flutter_pods(flutter_chat_application_path)
end
```

In the `Podfile` 's `post_install` block, call `flutter_post_install(installer)` , and with the statement of necessary permissions.

```
post_install do |installer|
flutter_post_install(installer) if defined?(flutter_post_install)
installer.pods_project.targets.each do |target|
   flutter_additional_ios_build_settings(target)
   target.build_configurations.each do |config|
         config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
           '$(inherited)',
           'PERMISSION_MICROPHONE=1',
           'PERMISSION_CAMERA=1',
           'PERMISSION_PHOTOS=1',
         ]
       end
 end
end
```

Run `pod install` .

Note:

When you change the Flutter plugin dependencies in `tencent_chat_module/pubspec.yaml` , run `flutter pub get` in your Flutter module directory to refresh the list of plugins read by the `podhelper.rb` script. Then, run pod install again from the root directory of your application.

You may need to run `arch -x86_64 pod install --repo-update` on the Mac with Apple Silicon, like M1 or M2.

The `podhelper.rb` script embeds your plugins, `Flutter.framework` , and `App.framework` into your project.

Option B - Embed frameworks in Xcode

Alternatively, you can generate the necessary frameworks and embed them in your application by manually editing your existing Xcode project.

You may do this if members of your team can't locally install Flutter SDK and CocoaPods, or if you don't want to use CocoaPods as a dependency manager in your existing applications.

You must run `flutter build ios-framework` every time you make code changes in your Flutter module.

It's recommended to use this option for the released version.

Steps:

Run the following command on your Flutter module.

The following example assumes that you want to generate the frameworks to `some/path/MyApp/Flutter/` .

```
flutter build ios-framework --output=some/path/MyApp/Flutter/
```

Embed and link the generated frameworks into your existing application in Xcode.

Adding Flutter Engines

It's recommended to add the Flutter module to your existing application.

You need to add the Chat and Call modules to your native app using Flutter engines. For details, see here.

There are two options for creating and managing Flutter engines: create a single `FlutterEngine` or create two `FlutterEngines` with `FlutterEngineGroup`.

Mode	Introduction	Pros	Cons	Sample Code
Single FlutterEngine	Both Chat and Call integrate into one `FlutterEngine`	Convenient	Since the Call module needs to automatically display the incoming call page when a call is received, it needs to be forced to redirect to the Flutter page, resulting in a poorer experience.	GitHub
Multiple FlutterEngines	The Chat and Call modules are located in two separate Flutter engines	The Call module exists independently in a Flutter engine, with independent page control. When a call comes in, the Flutter page is shown, which does not affect the page where the user is currently located, providing a better experience.	Minimize for the calling page is not allowed.	GitHub

In addition to the methods above, you can also integrate a native Chat SDK and Flutter SDK. For details, see here, and sample code can be found from GitHub.

Solution A: Multiple FlutterEngines (Recommended)

The advantage of using multiple Flutter instances is that each instance is independent and maintains its own internal navigation stack, UI, and application states. This simplifies the overall application code's responsibility for state

keeping and improves modularity. More details on the scenarios motivating the usage of multiple Flutters can be found at docs.flutter.dev/go/multiple-flutters.

The primary API for adding multiple Flutter instances on both Android and iOS is based on a new `FlutterEngineGroup` class to construct `FlutterEngine` s, rather than the `FlutterEngine` constructors used in the Solution B: Single FlutterEngine.
Whereas the `FlutterEngine` API was direct and easier to consume, the `FlutterEngine` spawned from the same `FlutterEngineGroup` have the performance advantage of sharing many of the common, reusable resources such as the GPU context, font metrics, and isolate group snapshot, leading to a faster initial rendering latency and lower memory footprint.
In our project, one single `FlutterEngineGroup` is used to manage the two `FlutterEngine` s, including Chat and Call modules.
You can refer to the sample code from GitHub repo this module.

Developing the Flutter Module

To embed Flutter into your existing application, you must first create a Flutter module.
From the command line, run:

```
cd some/path/
flutter create --template module tencent_chat_module
```

A Flutter module project is created at `some/path/tencent_chat_module/` . From that directory, you can run the same flutter commands you would in any other Flutter project, like `flutter run --debug` or `flutter build ios` . You can also run the module in Android Studio/IntelliJ or VS Code with the Flutter and Dart plugins.
This project contains a single-view example version of your module before it's embedded in your existing application, which is useful for incrementally testing the Flutter-only parts of your code.
The `tencent_chat_module` module directory structure is similar to a normal Flutter application:

```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
│   └── Flutter/podhelper.rb
├── lib/
│   └── main.dart
├── test/
└── pubspec.yaml
```

Now, you can add code within `lib/` .

The structure of `lib/`

Note:

The following structure and code are for demonstration purposes only. You can modify them to meet your actual needs.

Within `lib/`, you need to create three directories: call, chat, and common. These directories are used for the Call module, Chat module, and some common classes, respectively.

```
tencent_chat_module/
├── lib/
|   └── call/
|   └── chat/
|   └── common/
```

Common model classes

Create a new file named `common/common_model.dart`, as shown below. Create two classes in this file to define the communication standard between Flutter and the native application.

```
class ChatInfo {
String? sdkappid;
String? userSig;
String? userID;

ChatInfo.fromJSON(Map<String, dynamic> json) {
  sdkappid = json["sdkappid"].toString();
  userSig = json["userSig"].toString();
  userID = json["userID"].toString();
}

Map<String, String> toMap(){
  final Map<String, String> map = {};
  if(sdkappid != null){
    map["sdkappid"] = sdkappid!;
  }
  if(userSig != null){
    map["userSig"] = userSig!;
  }
  if(userID != null){
    map["userID"] = userID!;
  }
  return map;
}
}

class CallInfo{
```

```
String? userID;
String? groupID;

CallInfo();

CallInfo.fromJSON(Map<String, dynamic> json) {
  groupID = json["groupID"].toString();
  userID = json["userID"].toString();
}

Map<String, String> toMap(){
  final Map<String, String> map = {};
  if(userID != null){
    map["userID"] = userID!;
  }
  if(groupID != null){
    map["groupID"] = groupID!;
  }
  return map;
}
}
```

Chat Module

The following files and code are located in the `lib/chat` directory.

1. Create a file, model.dart, used as a state container.

This model is used to initialize and maintain the instance of Tencent Cloud Chat, offline line push module, global state, and the communication with native apps.

It's the core of the Chat module.

For detailed implementation, refer to the source code of the sample app, paying attention to the following three functions:

Future_handleMessage(MethodCall call): Listens for events from native app, including login and notification click events.

Future handleClickNotification(Map< String, dynamic> msg): The function invoked by the callback after clicking the notification.

Future initChat(): Initialize and log in Tencent Cloud Chat SDK and offline push plugin, upload token. This method uses the sync lock mechanism to ensure that only one can be executed at the same time, and after the initialization is successful, it will not be executed repeatedly.

Note:

Please configure the offline push before uploading the token and use this capability, see Integrating Offline Push.

2. Create a file, `chat_main.dart` . This is also used as the home page of the chat module.

Also, used as the home page of the chat module.

It shows the loading status before logged in, followed by the conversation list.
Besides, the current status of the application needs to be reported to the Tencent Cloud Chat backend upon each foreground/background switch from here.
Detailed implementation can refer to the sample code from GitHub repo.
2.1 Create a file, `push.dart` , used for maintaining the offline push plugin. Detailed implementation can refer to the sample code from GitHub repo.
2.2 Create a file, `conversation.dart` , used for implementing conversation list widget `TIMUIKitConversation` . Detailed implementation can refer to the sample code from GitHub repo.
2.3 Create a file, `user_profile.dart` , used to implement the user profile widget `TIMUIKitProfile` . Detailed implementation can refer to the sample code from GitHub repo.
2.4 Create a file, `group_profile.dart` , used to implement group profile widget `TIMUIKitGroupProfile` . Detailed implementation can refer to the sample code from GitHub repo.
2.5 Create a file, `chat.dart` , used for implementing the history message list and sending messages widget `TIMUIKitChat` . This page can also navigate to `user_profile.dart` and `conversation.dart` . Detailed implementation can refer to the sample code from GitHub repo.
At this point, the Chat module has been developed with the following structure:

```
tencent_chat_module/
├── lib/
│   └── call/
│       └── chat.dart
│       └── model.dart
│       └── chat_main.dart
│       └── push.dart
│       └── conversation.dart
│       └── user_profile.dart
│       └── group_profile.dart
│   └── chat/
│   └── common/
```

Call Module

This module is used for voice call and video call, provided by our calling plugin.
The key function of this module is to automatically show the call page by calling the native method when a new call invitation is received, as well as to accept the call request forwarded by the Chat module and actively initiate the call.
The following files and code are located in the `lib/call` directory.
1. Create a file, `model.dart` , used as a state container.
This model is used for initializing and maintaining the instance of Calling plug-in, global state, and the communication with native apps.
Is the core of the Call module.

Detailed implementation can refer to the source code of the sample app, while it's recommended to focus on these two functions:
_onRtcListener = TUICallingListener(...): The listener of the calling events, notify native to show this page, when receiving a new call.
Future_handleMessage(MethodCall call): The listener of the method channel call events, mainly used for initiating a call to others, when receiving the request from the Chat module, via native.
2. Create a file, `call_main.dart` , used as the main entry point of the Call module.
The `navigatorKey` used for launching the call page should be added here.
Detailed implementation can refer to the sample code from GitHub repo.

Configure the entry point for each module

After completing the three parts above, you can configure the entry point for each module, which serve as the entry for the Flutter engine.
1. Default entry
Open `lib/main.dart` , modify the default main functions to return an empty `MaterialApp` .
This method serves as the default entry point for the Flutter module. Under the management of FlutterEngineGroup in a multi-engine Flutter engine scenario, if no child Flutter engine sets any entry point, this method will not be used.
For example, in this tutorial, this default main() method is not used.

```
void main() {
WidgetsFlutterBinding.ensureInitialized();

runApp(MaterialApp(
  title: 'Flutter Demo',
  theme: ThemeData(
    primarySwatch: Colors.blue,
  ),
  home: Container(),
));
}
```

2. The entry for Chat module
Use `@pragma('vm:entry-point')` to mark a method as an entry point. The method named `chatMain` is the name of the entry.
In the native code, this name is also used to create the corresponding `FlutterEngine` .
Use the global `ChangeNotifierProvider` for state management to maintain `ChatInfoModel` data and business logic.

```
@pragma('vm:entry-point')
void chatMain() {
// This call ensures the Flutter binding has been set up before creating the
// MethodChannel-based model.
```

```
WidgetsFlutterBinding.ensureInitialized();

final model = ChatInfoModel();

runApp(
  ChangeNotifierProvider.value(
    value: model,
    child: const ChatAPP(),
  ),
);
}
```

3. The entry for Call module

This entry point is named as `callMain` .

Use global `ChangeNotifierProvider` status management to maintain `CallInfoModel` data and business logic.

```
@pragma('vm:entry-point')
void callMain() {
// This call ensures the Flutter binding has been set up before creating the
// MethodChannel-based model.
WidgetsFlutterBinding.ensureInitialized();

final model = CallInfoModel();

runApp(
  ChangeNotifierProvider.value(
    value: model,
    child: const CallAPP(),
  ),
);
}
```

At this point, the Dart code for the Flutter Module has been completed.

Now, let's take a look at the native integration for your existing app.

iOS Native development

In this section, `Swift` is used as an example, but `Objective-C` is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your iOS project within XCode.

If your existing application (MyApp) doesn't already have a `Podfile` , follow the CocoaPods getting started guide to add a Podfile to your project.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing iOS app.

FlutterEngineGroup

Create a `FlutterEngineGroup` to maintain and manage the `FlutterEngine` s.

The proper place to create a `FlutterEngineGroup` is specific to your host app. As an example, we demonstrate creating a `FlutterEngineGroup` , exposed as a property, on app startup in the app delegate.

Add the following to `AppDelegate.swift` .

```
@UIApplicationMain
class AppDelegate: FlutterAppDelegate {
lazy var flutterEngines = FlutterEngineGroup(name: "chat.flutter.tencent", project:
...
}
```

Create a singleton static object to hold `FlutterEngine` s.

This singleton is used for managing those `FlutterEngine` s in one place, and provides methods to the whole project to invoke methods related to the Flutter module.

In the sample code, a new navigator is used for the Chat ViewController. The Call ViewController is maintained dynamically with present and dismiss.

Create a new file named FlutterUtils.swift. For detailed code, refer to the sample code.

Mainly focus on:

private override init(): Initialize each Flutter instance, register method channel events.

func reportChatInfo(): Report the current user info to the Flutter module, for initialization and login Tencent Cloud Chat SDK.

func launchCallFunc(): Present the ViewController for Call module, invoked when new call income or user active it manually from Chat module.

func triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

Listen for and forward the notification click event

The initialization/token reporting/click event corresponding to the offline push notification is handled in the Flutter Chat module. Therefore, on the native side, only the ext of the click notification event needs to be passed through.

The reason we need to do this is because the clicking event has been consumed on the native side, so it is impossible for the Flutter Push plug-in to receive this event.

Add the following code to `AppDelegate.swift` .

At this point, the implementation for iOS is complete.

Android Native Development

Here, Kotlin is used as an example, but Java is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your Android project within Android Studio.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing Android app.

FlutterEngineGroup

Create a `FlutterEngineGroup` to maintain and manage the `FlutterEngine` s.

The proper place to create a `FlutterEngineGroup` is specific to your host app. As an example, we demonstrate creating a `FlutterEngineGroup` , exposed as a property, on app startup in the app delegate.

Create a new file, `FlutterUtils.kt` , and define a singleton static object `FlutterUtils` .

```
@SuppressLint("StaticFieldLeak")
object FlutterUtils {}
```

Create a `FlutterEngineGroup` to maintain and manage the `FlutterEngine` s.

Define a `FlutterEngineGroup` , `FlutterEngine` s and corresponding `MethodChannel` s in `FlutterUtils.kt` .

```
lateinit var context : Context
lateinit var flutterEngines: FlutterEngineGroup
private lateinit var chatFlutterEngine:FlutterEngine
private lateinit var callFlutterEngine:FlutterEngine

lateinit var chatMethodChannel: MethodChannel
lateinit var callMethodChannel: MethodChannel

// Initialize them
flutterEngines = FlutterEngineGroup(context)
...
```

Further developed for this singleton static object

The basic implementation logic of the sample app is that, using a new navigator for the `Activity` for both Chat and Chat.

The `Activity` for Chat is entered and exited by the user, while the `Activity` for Call has been entered and exited automatically, triggered by the listener or making a call manually.
Mainly focus on:
fun init(): Initialize each Flutter instance, register method channel events.
func reportChatInfo(): Reports the current user login info and SDKAppID to the Flutter module to initialize and log in to the Tencent Cloud Chat SDK.
fun launchCallFunc(): Present the `Activity` for Call module, invoked when new call income or user active it manually from Chat module.
fun triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.
You can refer to the sample app source code for this object.
Initialize the singleton static object above from the main entry `MyApplication`.
Transit the global context to the singleton static object, and initialize it from `MyApplication.kt`.

```
class MyApplication : MultiDexApplication() {

   override fun onCreate() {
       super.onCreate()
       FlutterUtils.context = this // new
       FlutterUtils.init()         // new
   }
}
```

Listen for and forward the notification click event
Only transit of the data of notification after clicking is necessary as, the initialization of Push plug-in, uploading token and the navigating for notification clicking events have been done in Flutter Chat module.
The reason why we need to do this is the clicking event has been consumed by Android Kotlin, so it is impossible for the Flutter Push plug-in to receive this event.
Note：
Due to the diversity and inconsistency among different manufacturers, we only take OPPO as an example. For the whole manufacturer's support, please refer to this documentation.
Add a new push certificate to the Tencent Cloud Chat console, Select **Open specified in-app page > activity** for the opening method and enter an activity to receive the notification clicking event with EXT data, it's suggested to set it as the home page or the main entrance. Like, we set `MainActivity` for our sample app, `com.tencent.chat.android.MainActivity`.

Add the following code to the Activity you set in the console in the previous step.

The EXT data of the notification can be found from `Bundle` when the `Activity` has been launched by the device, when the user clicks the notification.
You can receive the EXT from `Activity`, and transit them to Flutter.
You can refer to the source code of the sample app for this capability.

Now, we finished the implementation for Android.

Solution B: Single FlutterEngine

In this solution, the Chat module and Call module embed in one single Flutter instance.

As a result, those modules can only be shown or hidden at the same time.
You can refer to the sample code from GitHub repo this module.

Flutter Module development

To embed Flutter into your existing application, first create a Flutter module.
From the command line, run:

```
cd some/path/
flutter create --template module tencent_chat_module
```

A Flutter module project is created at `some/path/tencent_chat_module/`. From that directory, you can run the same flutter commands you would in any other Flutter project, like `flutter run --debug` or `flutter build ios`. You can also run the module in Android Studio/IntelliJ or VS Code with the Flutter and Dart plugins.
This project contains a single-view example version of your module before it's embedded in your existing application, which is useful for incrementally testing the Flutter-only parts of your code.
The `tencent_chat_module` module directory structure is similar to a normal Flutter application:

```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
│   └── Flutter/podhelper.rb
├── lib/
│   └── main.dart
├── test/
└── pubspec.yaml
```

Now, we can code within `lib/` .

main.dart

Modify `main.dart` , integrating TUIKit, Offline Push plug-in and Call Plug-in.
The global state, method channel and our Tencent Cloud Chat SDKs, maintained by `ChatInfoModel` .
After receiving the login user info from Native, invoke `_coreInstance.init()` and `_coreInstance.login()` to initialize and login the SDK. Also, Call plug-in and Push plug-in need to be initialized.
Note:
Please configure the offline push before uploading the token and use this capability, referring to this documentation.
Tips for Call plug-in:
When a new call invitation is received, call the native method to check if the user is currently on the Flutter page. If not, force the page to redirect to this module to display the incoming call page.
Tips for Push plug-in:
The callback event of notification clicking is passed from the native layer and used to navigate to the corresponding chat from EXT data.
Also, this is used as the home page of the chat module. It shows the loading status before logged in, followed by the conversation list.
In addition, the current status of the application needs to be reported to the Tencent Cloud Chat backend upon each foreground/background switch from here. For details, refer to this document.
Detailed implementation can refer to the sample code from GitHub repo.

Other widgets from TUIKit

1. Create a file, `push.dart` , used for maintaining the offline push plugin. Detailed implementation can refer to the sample code from GitHub repo.
2. Create a file, `conversation.dart` , used to implement group profile widget `TIMUIKitGroupProfile` . Detailed implementation can refer to the sample code from GitHub repo.
3. Create a file, `user_profile.dart` , used to implement the user profile widget `TIMUIKitProfile` . Detailed implementation can refer to the sample code from GitHub repo.
4. Create a file, `group_profile.dart` , used to implement group profile widget `TIMUIKitGroupProfile` . Detailed implementation can refer to the sample code from GitHub repo.
5. Create a file, `chat.dart` , used for implementing the history message list and sending messages widget `TIMUIKitChat` . This page can also navigate to `user_profile.dart` and `conversation.dart` .Detailed implementation can refer to the sample code from GitHub repo.

At this point, the Flutter module has been developed.

iOS Native development

Here, we take `Swift` as an example, while `Objective-C` is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your iOS project within XCode.

If your existing application (MyApp) doesn't already have a `Podfile`, follow the CocoaPods getting started guide to add a Podfile to your project.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing iOS app.

FlutterEngine

Create a FlutterEngine.

The proper place to create a `FlutterEngine` is specific to your host app. As an example, we demonstrate creating a `FlutterEngine`, exposed as a property, on app startup in the app delegate.

```
import UIKit
import Flutter
import FlutterPluginRegistrant

@UIApplicationMain
class AppDelegate: FlutterAppDelegate { // More on the FlutterAppDelegate.
 lazy var flutterEngine = FlutterEngine(name: "tencent cloud chat")

 override func application(_ application: UIApplication, didFinishLaunchingWithOpti
   // Runs the default Dart entry point with a default Flutter route.
   flutterEngine.run();
   GeneratedPluginRegistrant.register(with: self.flutterEngine);
   return super.application(application, didFinishLaunchingWithOptions: launchOptio
 }
}
```

Create a singleton static object to manage the FlutterEngine.

This singleton is used for managing `FlutterEngine` in one place, and provides methods to the whole project to invoke methods related to the Flutter module.

The basic implementation logic of the sample app is that, using a new navigator for the ViewController of Flutter module, and show or hidden can be handled automatically according to call.

Create a new file, `FlutterUtils.swift`, and coding, refer to our sample app source code.

Mainly focus on:

private override init(): Initializes each Flutter instance, registers method channel, and listens for events.

func reportChatInfo(): Report the current user info to the Flutter module, for initialization and login Tencent Cloud Chat SDK.

func launchChatFunc(): Present the ViewController for Flutter module.

func triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

Listen for and forward the notification click event

Only transit of the data of notification after clicking is necessary as, the initialization of Push plug-in, uploading token and the navigating for notification clicking events have been done in Flutter Chat module.

The reason why we need to do this is the clicking event has been consumed by iOS Swift, so it is impossible for the Flutter Push plug-in to receive this event.

Add the following codes to `AppDelegate.swift` .

At this point, the implementation for iOS is complete.

Android Native Development

Here, we take `Kotlin` as an example, while `Java` is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your Android project within Android Studio.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing Android app.

FlutterEngine

Create a singleton static object to manage the FlutterEngine.

This singleton is used for managing `FlutterEngine` in one place, and provides methods to the whole project to invoke methods related to the Flutter module.

Create a new file, `FlutterUtils.kt` , and define a singleton static object `FlutterUtils` .

```
@SuppressLint("StaticFieldLeak")
object FlutterUtils {}
```

Create a `FlutterEngine` .

Define a `FlutterEngine` and corresponding `MethodChannel` in `FlutterUtils.kt` .

```
lateinit var context : Context
private lateinit var flutterEngine:FlutterEngine
```

```
flutterEngine = FlutterEngine(context)
```

Further developed for this singleton static object

The basic implementation logic of the sample app is that, using a new navigator for the Activity for both Chat and Chat.

Mainly focus on:

fun init(): Initialize each Flutter instance, register method channel events.

func reportChatInfo(): Reports the current user login info and SDKAppID to the Flutter module to initialize and log in to the Tencent Cloud Chat SDK.

fun launchChatFunc(): Present the `Activity` for Flutter module.

fun triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

Detailed implementation can refer to the sample code from GitHub repo.

Initialize the singleton static object above from the main entry `MyApplication`.

Transit the global context to the singleton static object, and initialize it from `MyApplication.kt`.

```
class MyApplication : MultiDexApplication() {

   override fun onCreate() {
       super.onCreate()
       FlutterUtils.context = this // new
       FlutterUtils.init()         // new
   }
}
```

Listen for and forward the notification click event

Only transit of the data of notification after clicking is necessary as, the initialization of Push plug-in, uploading token and the navigating for notification clicking events have been done in Flutter Chat module.

The reason why we need to do this is the clicking event has been consumed by Android Kotlin, so it is impossible for the Flutter Push plug-in to receive this event.

> Due to the diversity and inconsistency among different manufacturers, we only take OPPO as an example. For the whole manufacturer's support, please refer to this documentation.

Add a new push certificate to the Tencent Cloud Chat console, Select **Open specified in-app page > activity** for the opening method and enter an activity to receive the notification clicking event with EXT data, it's suggested to set it

as the home page or the main entrance. Like, we set `MainActivity` for our demo, `com.tencent.chat.android.MainActivity`.

Add the following code to the Activity you set in the console in the previous step.
The EXT data of the notification can be found from `Bundle` when the `Activity` has been launched by the device, when the user clicks the notification.
You can receive the EXT from `Activity`, and transit them to Flutter.
You can refer to the demo source code for this capability.

At this point, the implementation for Android is complete.

Additional solution: Initialize Tencent Cloud Chat from Native SDK

Sometimes, you may prefer to integrate a chat module to your existing UI without a complex chat page.
For example, during a game, you want to let players chat with each other during the match without navigating to the full screen chat page.
Means, you may not wish to launch a complex Flutter engine, before the user switches to the chat page, but hope they can still chat in a small module directly.
In this case, you can initialize and login using the native SDK and build in-app chat features.
Note:
However, you can also choose to initialize and login within Flutter up to your needs. This process should only be executed once, no matter where you execute it.
It's unnecessary to import Native SDK manually, as our Flutter SDK can help you integrate it.

Initialize and login

iOS Swift is used as an example to demonstrate how to initialize and log in with the native SDK.

```
import ImSDK_Plus


func initTencentChat(){
       if(isLoginSuccess == true){
           return
       }
       let data = V2TIMManager.sharedInstance().initSDK( Yours SDKAPPID , config: n
```

```
        if (data == true){
            V2TIMManager.sharedInstance().login(
                chatInfo.userID,
                userSig: chatInfo.userSig,
                succ: {
                    self.isLoginSuccess = true
                    self.reportChatInfo()
                },
                fail: onLoginFailed()
            )
        }
}
```

After that, you could use the API provided by Native SDK to implement your chat modules to your existing UI page manually.

For more information about the Native SDK, please refer to this documentation.

Initialize Flutter TUIKit

After initialization and login from the native SDK, the user info should be provided to Flutter TUIKit by invoking _coreInstance.setDataFromNative().

```
final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
_coreInstance.setDataFromNative(userId: chatInfo?.userID ?? "");
```

You can refer to the sample code from GitHub repo this module.

This completes the tutorial for using hybrid development with Flutter to integrate Tencent Cloud Chat to your existing app.

If you have any questions, feel free to contact us.

Telegram Group

WhatsApp Group

Reference

1.1 Integrate a Flutter module into your Android project.

1.2 Integrate a Flutter module into your iOS project.

1.3 Adding a Flutter screen to an iOS app.

1.4 Multiple Flutter screens or views.