

eKYC

Integration Guide

Product Documentation



Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

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

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

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

Contents

Integration Guide

Getting Started

Integrating Liveness Detection and Face Comparison (Mobile HTML5)

Integration Process

HTML5 Compatibility and Mode Switch Description

Integrating Liveness Detection and Face Comparison (App SDK)

Integration Process

SDK API Description

Android API Description

iOS API Description

SDK Custom Capabilities

Android Custom Capabilities

iOS Custom Capabilities

FAQs

Integrating Identity Verification (App SDK)

Integration Process

SDK API Description

APIs for Android

APIs for iOS

FAQs

Integrating Identity Verification (Mobile HTML5)

Integration Process

Liveness Detection and Face Comparison (Pure API)

Integration Process

Other Guide

Quick API Run

Connecting to TencentCloud API

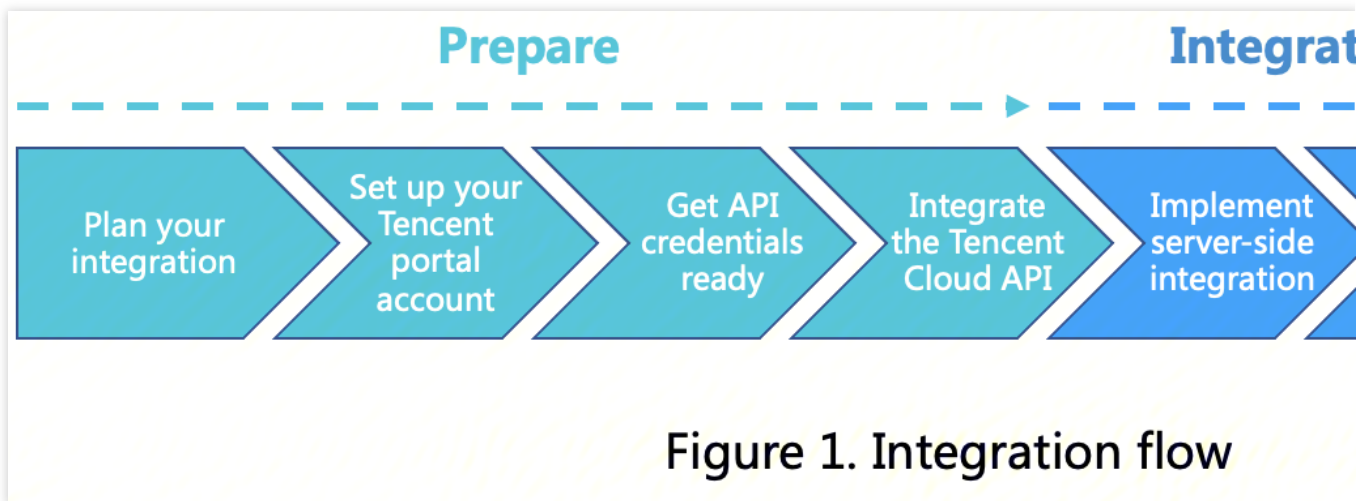
Error Codes

Integration Guide

Getting Started

Last updated : 2023-12-28 15:58:32

Integrate the Tencent Cloud eKYC service with your application in three stages: prepare, integration, and test.



The following sections describe the three stages in detail.

Prepare

Prepare for the integration in the following steps:

1. Plan your integration

Select appropriate products based on the features you need.

Subproduct	Feature Overview	Feature Details	Supported Country/Region	Supported Integration Method
Liveness detection and face comparison	Determines whether the user in the video is a live person and compares with the photo provided by the backend to determine whether they are the same person.	<ul style="list-style-type: none">●Real-time face detection●Liveness detection (available modes: video-based, motion-based, reflection-based, and blinking-and-reflection-based)	All regions	<ul style="list-style-type: none">●App SDK●Mobile HTML5●Pure API

		●Comparison with the photo provided by the backend		
Identity verification	Provides an automated identity authentication solution that prompts users to take a photo of their identity documents and take video selfies, so as to determine whether a user is a live person and whether the user is the person shown in the document.	<ul style="list-style-type: none"> ●ID document OCR ●ID photo extraction ●Real-time face detection ●Liveness detection (available modes: video-based, motion-based, reflection-based, and blinking-and-reflection-based) ●ID photo and selfie comparison 	<ul style="list-style-type: none"> ●Hong Kong (China) ●Malaysia ●The Philippines ●Indonesia ●Singapore (If you need to support identity verification in other regions, please contact us.) 	App SDK

Tencent Cloud eKYC provides four methods for you to integrate with different products as needed.

A. Liveness detection and face comparison

Liveness detection and face comparison (App SDK): Use this method if you want to integrate eKYC's liveness detection and face comparison service with your mobile application (iOS/Android).

Liveness detection and face comparison (mobile HTML5): Use this method if you want to integrate eKYC's liveness detection and face comparison service with a mobile web (HTML5) application or interact with it through a modern mobile web browser.

Liveness detection and face comparison (pure API): Use this method if you want to integrate eKYC's liveness detection and face comparison service with your server-side application via API.

B. Identity verification

Identity Verification (App SDK): Use this method if you want to integrate eKYC's identity verification service with your mobile application (iOS/Android).

2. Set up your Tencent portal account

A. Sign up for a Tencent Cloud account

To use the eKYC service, sign up for a Tencent Cloud account as instructed in [Signing Up](#) and complete [Enterprise Identity Verification](#). (If you have already signed up, skip this step.)

B. Activate the eKYC service

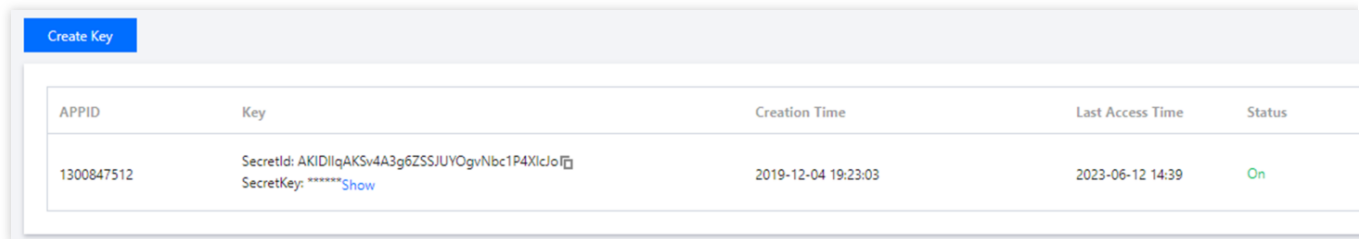
Log in to the [eKYC console](#) using your Tencent Cloud account and activate the service.

3. Get API credentials ready

Tencent Cloud uses `SecretId` and `SecretKey` to verify your identity and permissions. You can get the Tencent Cloud API credentials in the following steps:

A. Go to the [TencentCloud API key management](#) page and select **CAM > API Key Management** on the left sidebar to enter the API key management page.

B. Click **Create Key** to create a key and save the `SecretId` and `SecretKey` for subsequent API calls. You can skip this step if you already have a Tencent Cloud key.



Create Key				
APPID	Key	Creation Time	Last Access Time	Status
1300847512	SecretId: AKIDllqAKSv4A3g6ZSSJUYOgvNbc1P4XlcJoFj SecretKey: ***** Show	2019-12-04 19:23:03	2023-06-12 14:39	On

4. Integrate the Tencent Cloud API

Tencent Cloud APIs can be called on the online debugging page of API 3.0 Explorer. To view the input parameters and responses of APIs, see [Quick API Run](<https://www.tencentcloud.com/document/product/1061/3702914517d180223e7d6a5f03a5868ed28311>). Before integrating Tencent Cloud API, you must integrate Tencent Cloud SDK to simplify the API integration process as well as to ensure that both API requests and responses meet expectations. For more information about integration, see [Connecting to TencentCloud API](#).

Integration

eKYC provides different integration methods. Complete integration by following the instructions provided in the guide corresponding to your integration method.

Integration guide:

[Integrating Liveness Detection and Face Comparison \(App SDK\)](#)

[Integrating Liveness Detection and Face Comparison \(Mobile HTML5\)](#)

[Liveness Detection and Face Comparison \(Pure API\) APIs](#)

[Integrating Identity Verification \(App SDK\)](#)

Test

After completing integration, you can verify it through testing. We strongly recommend you complete full testing before use in the production environment. As Tencent Cloud does not provide a test environment, you can complete your

integration testing in a Tencent Cloud production environment.

Integrating Liveness Detection and Face Comparison (Mobile HTML5)

Integration Process

Last updated : 2024-01-25 10:37:31

This document introduces the overall integration process of eKYC liveness detection and face comparison (mobile HTML5).

Preparations

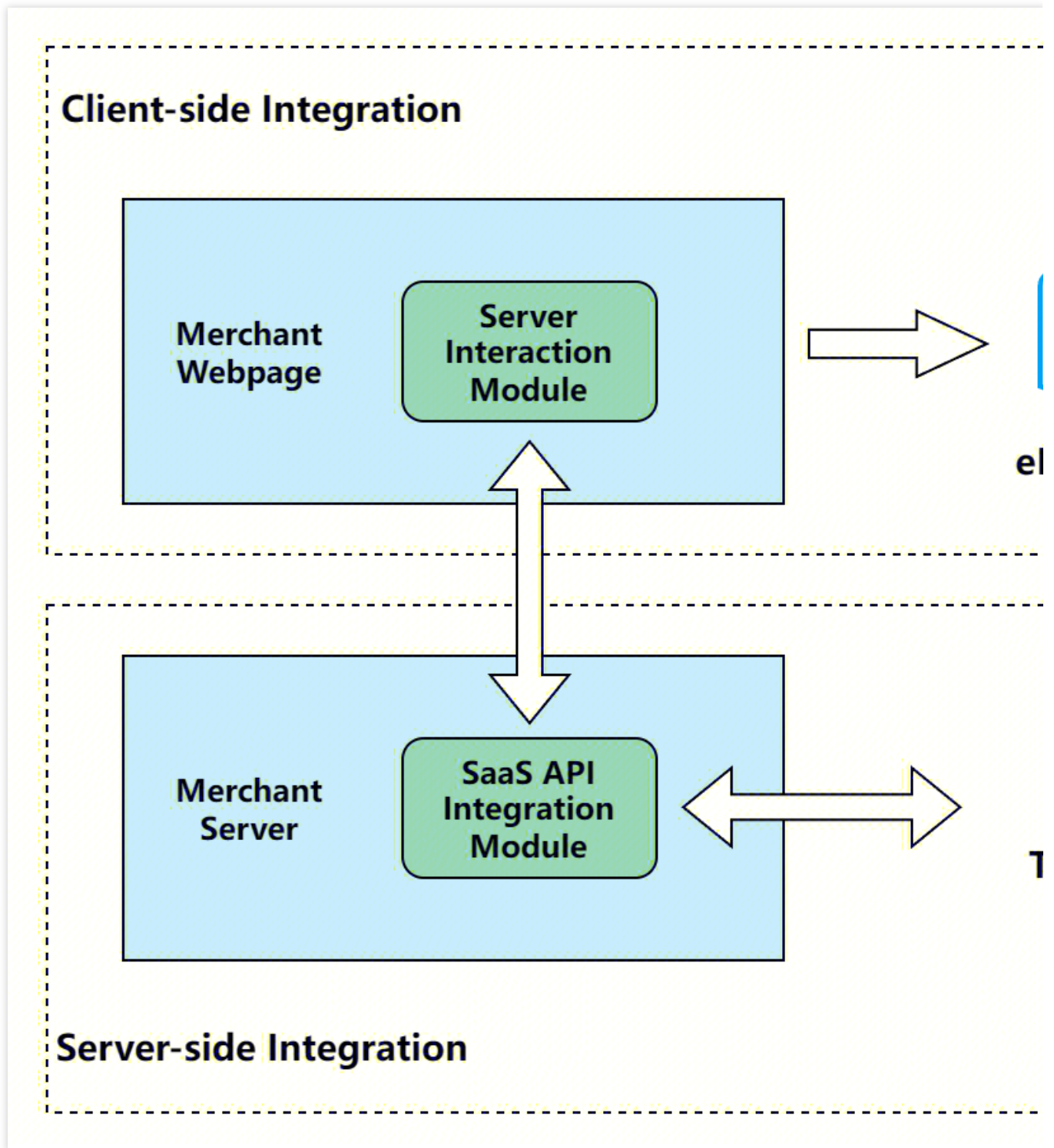
Sign up for a Tencent Cloud account. For more information, see [Signing Up](#).

Complete enterprise identity verification. For more information, see [Enterprise Identity Verification Guide](#).

Log in to the [eKYC console](#).

Overall Architecture

The following figure shows the architecture of eKYC liveness detection and face comparison (mobile HTML5).



Overall Interaction Process

The following figure shows the overall interaction logic between a user and Tencent Cloud eKYC. The roles involved are described as follows:

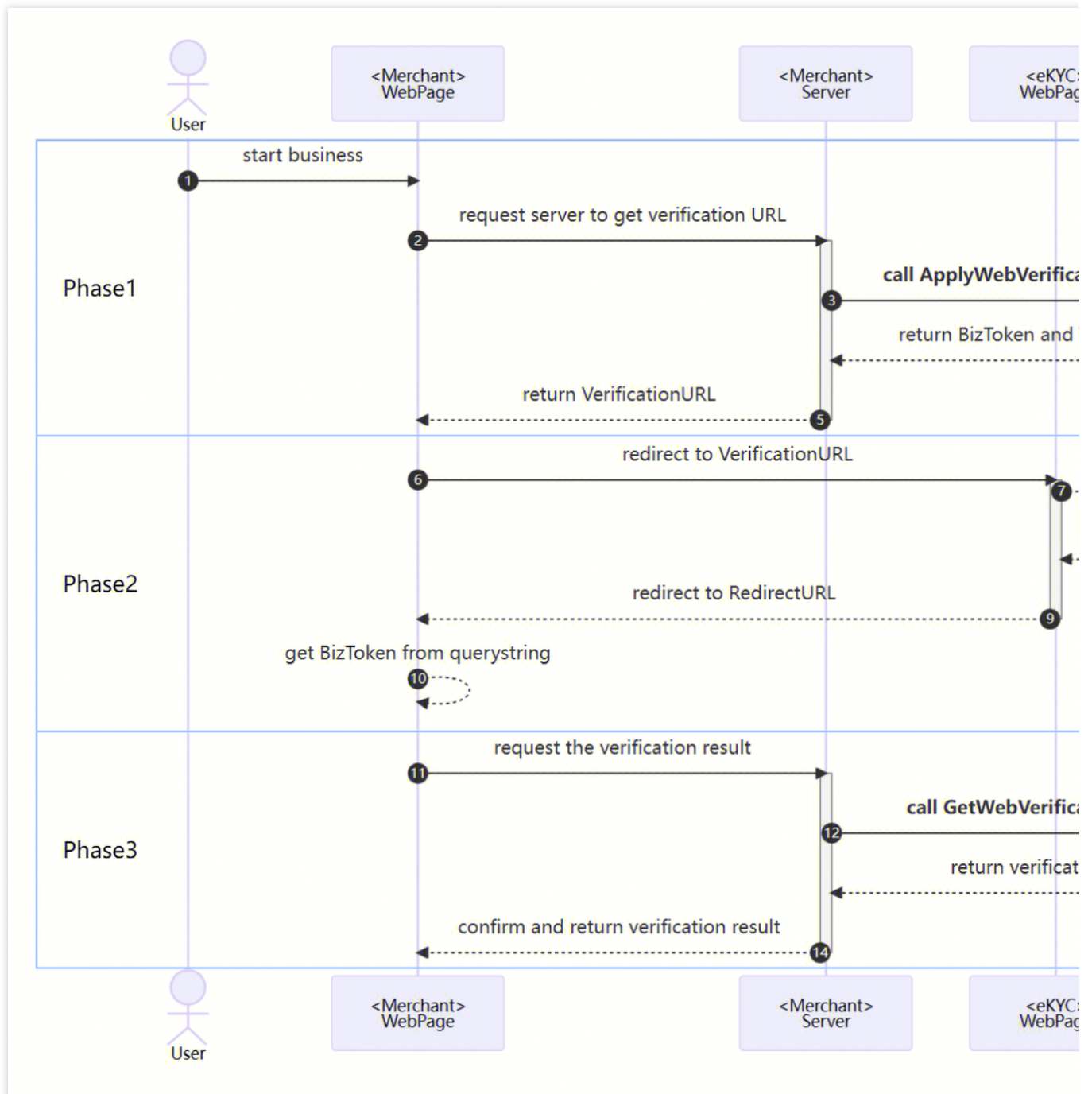
User: Mobile HTML5 client.

Merchant WebPage: Merchant's frontend page.

Merchant Server: Merchant's backend service.

eKYC WebPage: eKYC frontend page.

eKYC Server: eKYC backend service.



The recommended detailed interaction process is as follows:

Phase 1

1. A user starts a business, which triggers the verification process.
2. The Merchant WebPage sends a request to the Merchant Server, informing it to initiate a verification process.
3. The Merchant Server calls the [ApplyWebVerificationBizTokenIntl](#) API, passing in relevant parameters. For more information, see the description of step 1 in "Server-Side Integration".

4. After receiving the request, the eKYC Server returns the BizToken and VerificationURL for the verification process to the Merchant Server.
5. The Merchant Server stores the obtained BizToken and sends the VerificationURL to the Merchant WebPage.

Phase 2

6. The Merchant WebPage redirects to the VerificationURL to open the eKYC WebPage. For more information, see the description of step 1 in "Client-Side Integration".
7. The user completes the verification process on the eKYC WebPage.
8. After the verification is completed, the eKYC Server sends the verification result to the eKYC WebPage, and the Merchant WebPage displays the result page.
9. After the user taps "Next", the eKYC WebPage redirects back to the RedirectURL, with the token parameter added.
10. The Merchant WebPage obtains the token parameter for the current verification process from the URL. For more information, see the description of step 2 in "Client-Side Integration".

Phase 3

11. The Merchant WebPage sends a request to the Merchant Server, informing it to obtain the verification result.
12. The Merchant Server calls the [GetWebVerificationResultIntl](#) API, passing in relevant parameters. For more information, see the description of step 2 in "Server-Side Integration".
13. After receiving the request, the eKYC Server returns the detailed information of the verification process to the Merchant Server.
14. The Merchant Server sends the result back to the Merchant WebPage, which then proceeds with the subsequent business process based on the result.

Server-Side Integration

1. Generate the verification URL (corresponding to Phase 1)

Call `ApplyWebVerificationBizTokenIntl` to obtain the values of **BizToken** and **VerificationURL**. This corresponds to step 3 in the interaction process.

CompareImageBase64: The Base64-encoded string (max 8 MB in size) of the photo to be compared.

RedirectURL: Set this parameter to the web callback URL to redirect to after the verification is completed.

`RedirectURL` should include the protocol, hostname, and path. An example value is

`https://www.tencentcloud.com/products/faceid` . After the verification process is completed, the

`BizToken` of this process will be added to the callback URL in the format of

`https://www.tencentcloud.com/products/faceid?token={BizToken}` before redirect.

Extra: The passthrough parameter of the business, max 1,000 characters, which will be returned in

`GetWebVerificationResultIntl` .

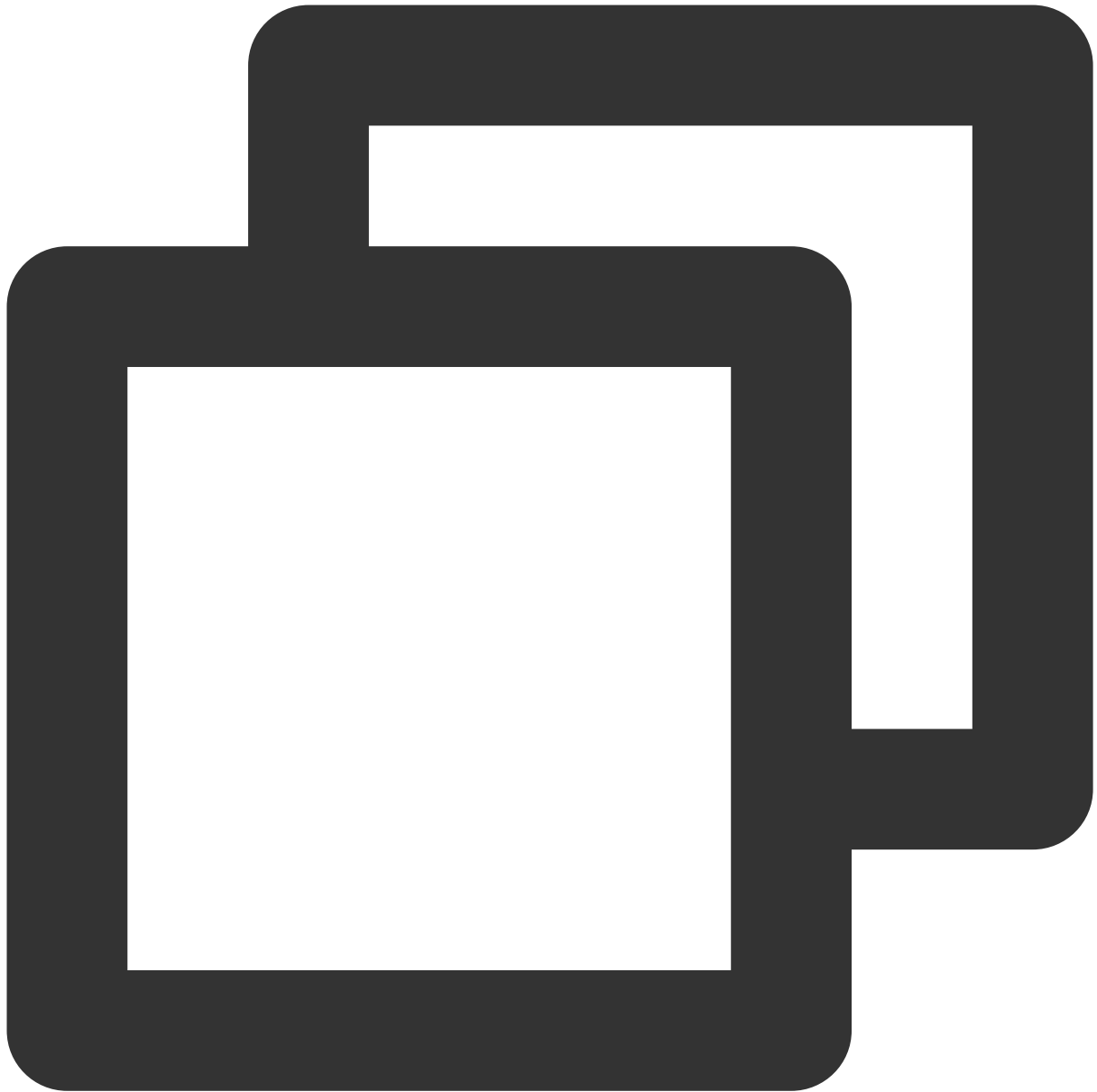
Config: Custom configuration for the verification page. This is optional.

The data structure of `Config` is as follows:

AutoSkip: When verification is successful, whether to skip the result display page and automatically redirect to

`RedirectURL` . The default value is `false` .

Sample API call:



```
package main

import (
    "fmt"
    "os"

    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/profile"
    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/regions"
    faceid "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/faceid/v"
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/errors"
```

```
)

func ApplyWebVerificationBizTokenIntl(imageBase64 string) {
    // Set TencentCloud API access key
    credential := common.NewCredential(
        os.Getenv("TENCENTCLOUD_SECRET_ID"),
        os.Getenv("TENCENTCLOUD_SECRET_KEY"),
    )
    cpf := profile.NewClientProfile()
    client, _ := faceid.NewClient(credential, regions.Singapore, cpf)
    request := faceid.NewApplyWebVerificationBizTokenIntlRequest()
    request.RedirectURL = common.StringPtr("https://www.tencentcloud.com/products/fac")
    // Pass in CompareImageBase64 and Extra
    request.CompareImageBase64 = common.StringPtr(imageBase64)
    request.Extra = common.StringPtr("ExtraString")
    response, err := client.ApplyWebVerificationBizTokenIntl(request)
    if _, ok := err.(*errors.TencentCloudSDKError); ok {
        fmt.Printf("An API error has returned: %s", err)
        return
    }
    if err != nil {
        panic(err)
    }
    // Obtain BizToken and VerificationURL
    bizToken := *response.Response.BizToken
    verificationURL := *response.Response.VerificationURL
    fmt.Printf("BizToken: %s, VerificationURL: %s", bizToken, verificationURL)
}
```

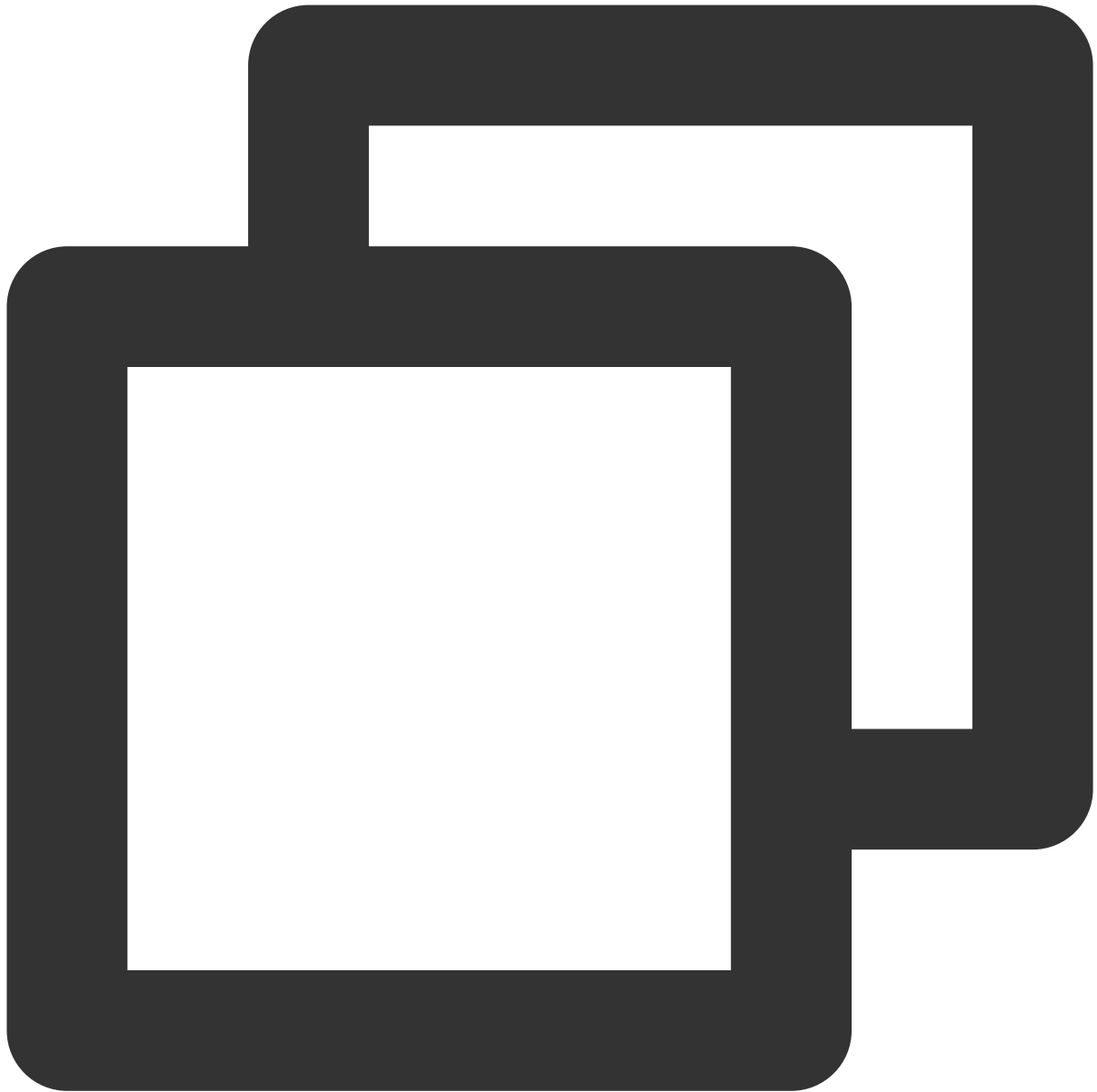
2. Check the verification result (corresponding to Phase 3)

After the verification process is completed, the merchant frontend requests the merchant server to obtain the verification result. The merchant server then calls the [GetWebVerificationResultIntl](#) API to obtain the final result and returns it to the frontend page. This corresponds to step 12 in the interaction process.

The final verification result is subject to the information returned by this API. When `ErrorCode` in the response is 0, it indicates that the verification is successful. In other cases, verification failed. For other error codes, see [Liveness Detection and Face Comparison \(Mobile HTML5\) Error Codes](#).

BizToken: The BizToken generated by the `ApplyWebVerificationBizTokenIntl` API, which is a unique identifier for the current verification process.

Sample API call:



```
package main

import (
    "fmt"
    "os"

    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/profile"
    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/regions"
    faceid "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/faceid/v"
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/errors"
```

```
)

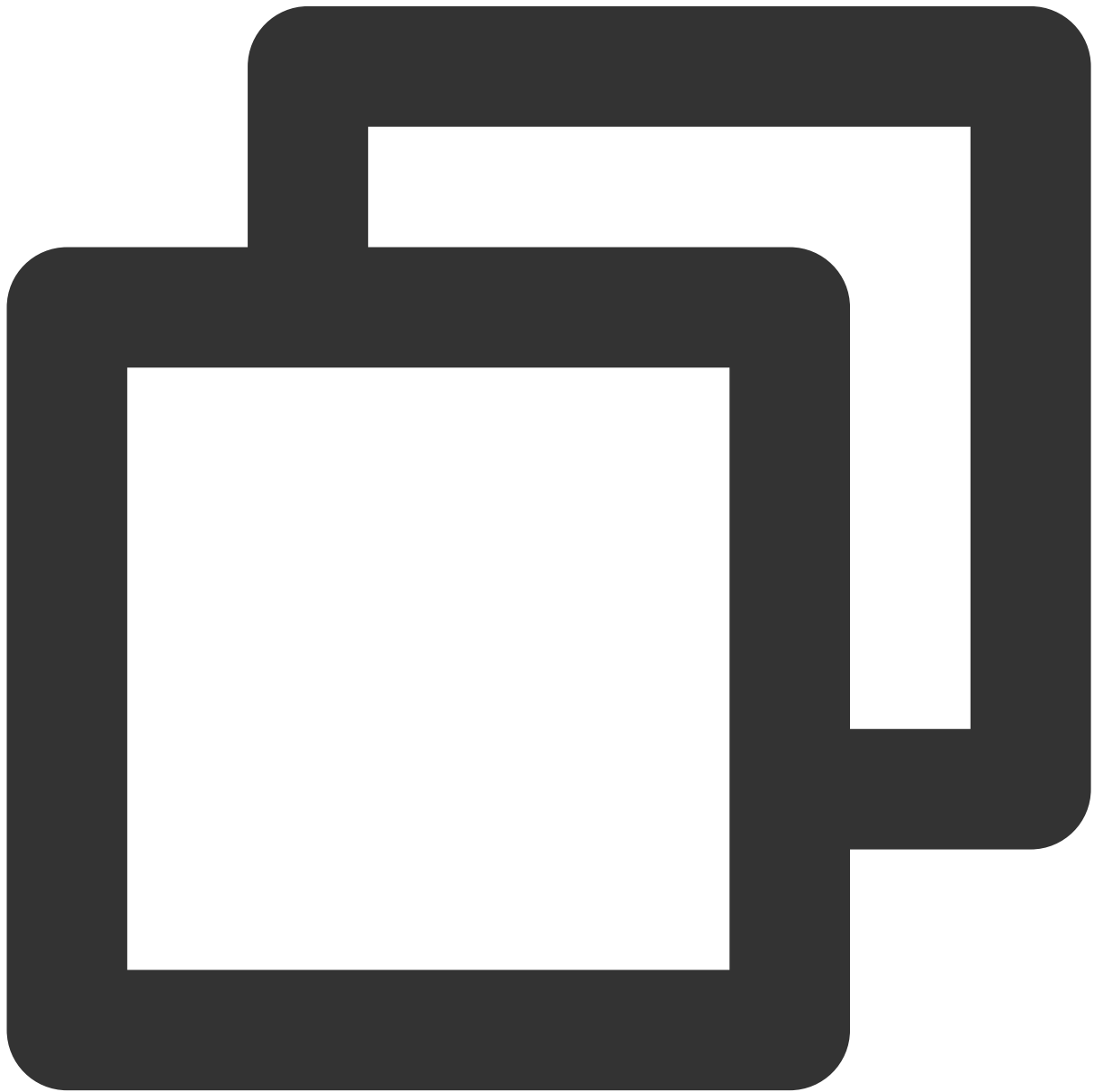
func GetWebVerificationResult(bizToken string) {
    // Set TencentCloud API access key
    credential := common.NewCredential(
        os.Getenv("TENCENTCLOUD_SECRET_ID"),
        os.Getenv("TENCENTCLOUD_SECRET_KEY"),
    )
    cpf := profile.NewClientProfile()
    client, _ := faceid.NewClient(credential, regions.Singapore, cpf)
    request := faceid.NewGetWebVerificationResultIntlRequest()
    // Pass in BizToken
    request.BizToken = common.StringPtr(bizToken)
    response, err := client.GetWebVerificationResultIntl(request)
    if _, ok := err.(*errors.TencentCloudSDKError); ok {
        fmt.Printf("An API error has returned: %s", err)
        return
    }
    if err != nil {
        panic(err)
    }
    if response.Response.ErrorCode == nil {
        fmt.Print("the verification is uncompleted.")
        return
    }
    errorCode := *response.Response.ErrorCode
    errorMsg := *response.Response.ErrorMsg
    if errorCode == 0 {
        // Verification succeeded
        fmt.Print("Success")
    } else {
        // Verification failed
        fmt.Printf("Fail: %s\\n", errorMsg)
    }
}
```

Client-Side Integration

1. Use VerificationURL to initiate verification (corresponding to Phase 2)

The client frontend page redirects to the VerificationURL received from the server to initiate the verification process. The user completes the liveness detection and face comparison process as prompted. This corresponds to step 6 in the interaction process.

Sample code:



```
// Obtain VerificationURL from the server
const VerificationURL = 'https://sg.faceid.qq.com/reflect/?token=*****';
// Redirect the frontend page
window.location.href = VerificationURL;
```

2. Obtain BizToken from the callback address and request the verification result from the backend (corresponding to Phase 2)

After the verification is completed, the page is redirected to `RedirectURL`. The BizToken parameter for the current process is added to the RedirectURL. By parsing the RedirectURL, you can obtain the BizToken parameter, which is used to obtain the verification result. This corresponds to step 12 in the interaction process.

Sample code:



```
// Obtain RedirectURL
const RedirectURL = "https://*?token={BizToken}";

// Parse RedirectURL to obtain the BizToken parameter, which is used to obtain the
const bizToken = getURLParameter(RedirectURL, "token");
if (bizToken) {
    // Use bizToken to obtain the verification result
}

/**
 * Get URL parameters
```



```
/* @params url The URL to be queried
/* @params variable The parameter to be queried
*/
function getUrlParameter(url, variable) {
  const query = url.split('?')[1] || '';
  const vars = query.split('&');
  for (let i = 0; i < vars.length; i++) {
    const pair = vars[i].split('=');
    if (pair[0] == variable) {
      return pair[1];
    }
  }
  return (false);
}
```

HTML5 Compatibility and Mode Switch

Description

Last updated : 2023-07-11 11:54:15

Compatibility description

The web real-time communication technology has the following compatibility requirements for browsers and mobile operating systems:

Mobile OS	Browser	Compatibility Requirements
iOS	Browser built in WeChat	iOS 14.3+ and WeChat 6.5+
	Safari	iOS 11.1.2+ and Safari 11+
	Chrome	iOS 14.3+
Android	Browser built in WeChat	Supported
	Built-in browsers	Android 7+, with great compatibility with the built-in browsers of Huawei, OPPO, vivo, MEIZU, Honor, and Samsung (80% supported) and average compatibility with the built-in browser of Xiaomi (30% supported)
	Other browsers	Android 7+, where Chrome is supported, but QQ Browser and UC Browser are not

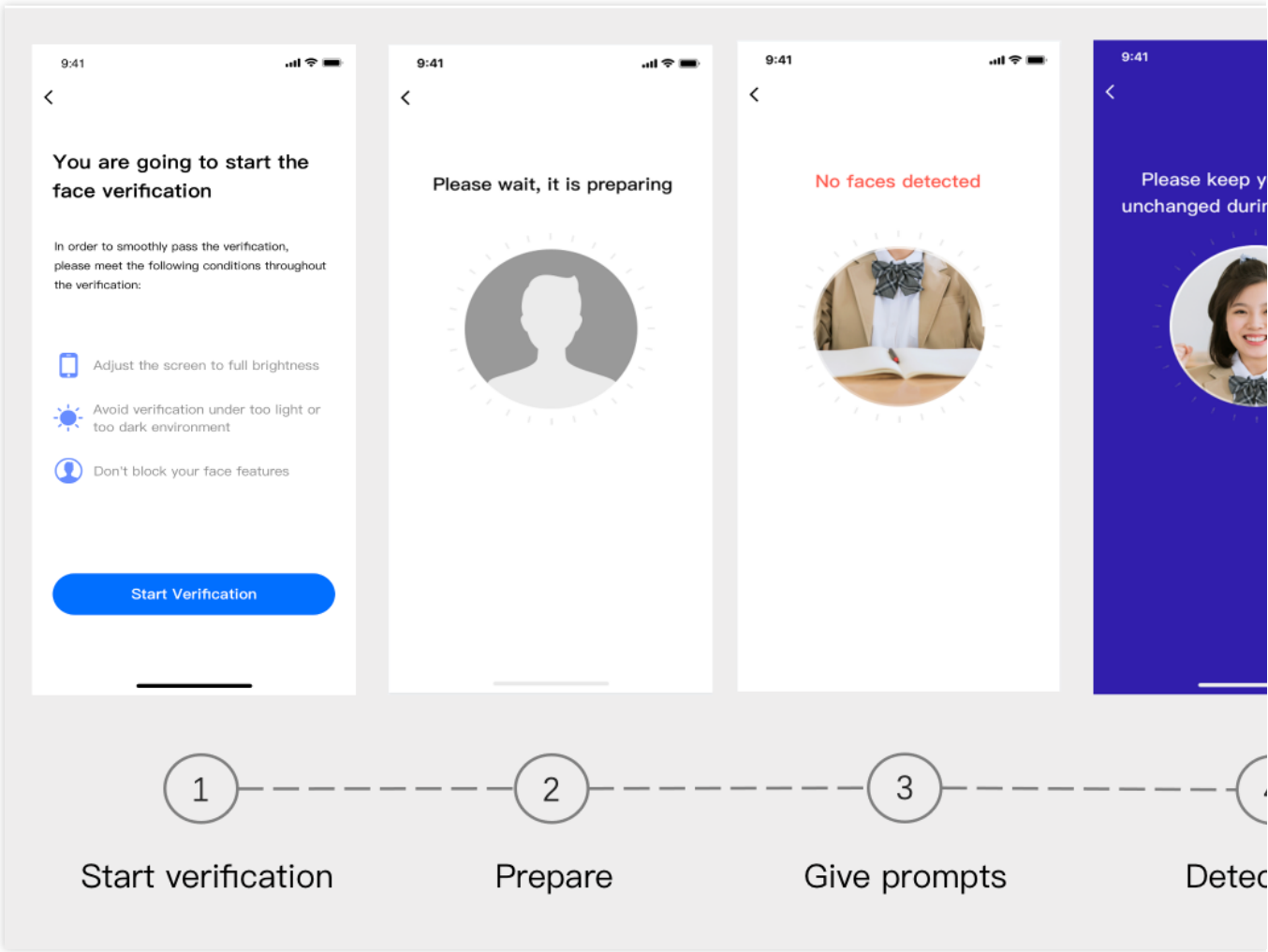
NOTE :

1. Due to H.264 copyright restrictions, Chrome and Chrome WebView-based browsers on Huawei devices don't support the real-time communication technology.
2. Under circumstances where the real-time communication technology isn't supported, web face recognition will switch from reflection-based liveness detection to video recording mode, so as to ensure that the user can properly complete the identity verification process.

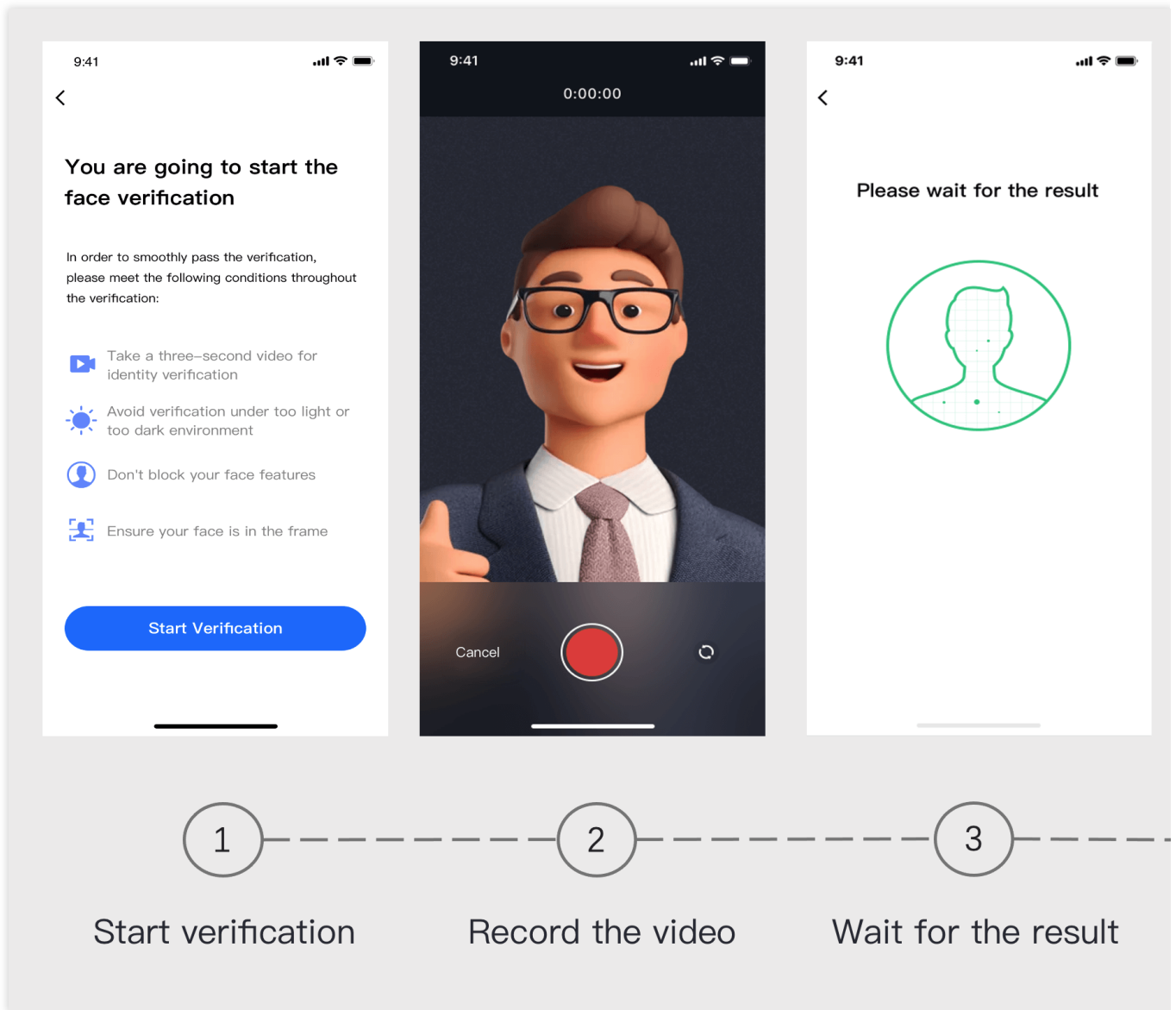
Mode switch description

In the web face recognition process, reflection-based liveness detection mode will be used first. However, if the real-time communication compatibility requirements cannot be met, the process will automatically switch to video-based liveness detection mode. The service processes in the two modes are as follows:

Web face recognition process in reflection-based liveness detection mode:



Web face recognition process in video-based liveness detection mode:



Camera access description for reflection-based liveness detection

When you call the web face recognition service, reflection-based liveness detection mode requires the user's camera access. If the user denies the access request, the face recognition and access request process needs to be entered again. Certain browsers cannot pull the access granting page; in this case, the user can try clearing the browser's cache.

Returned Message	Action
Unable to access your camera/mic. Please make sure that there is no other app requesting access to them and try again.	Advise the user to check whether the required camera is in use.
Mic and camera permissions of your device are required during the whole verification. Please clear your browser cache and try again.	Advise the user to enter again and grant the camera access.
Please check whether the camera/mic can be accessed normally and	Advise the user to check whether the

try again

required camera works properly.

Integrating Liveness Detection and Face Comparison (App SDK)

Integration Process

Last updated : 2024-01-19 14:57:08

Integration Preparations

Sign up for a Tencent Cloud account. For more information, see [Signing Up](#).

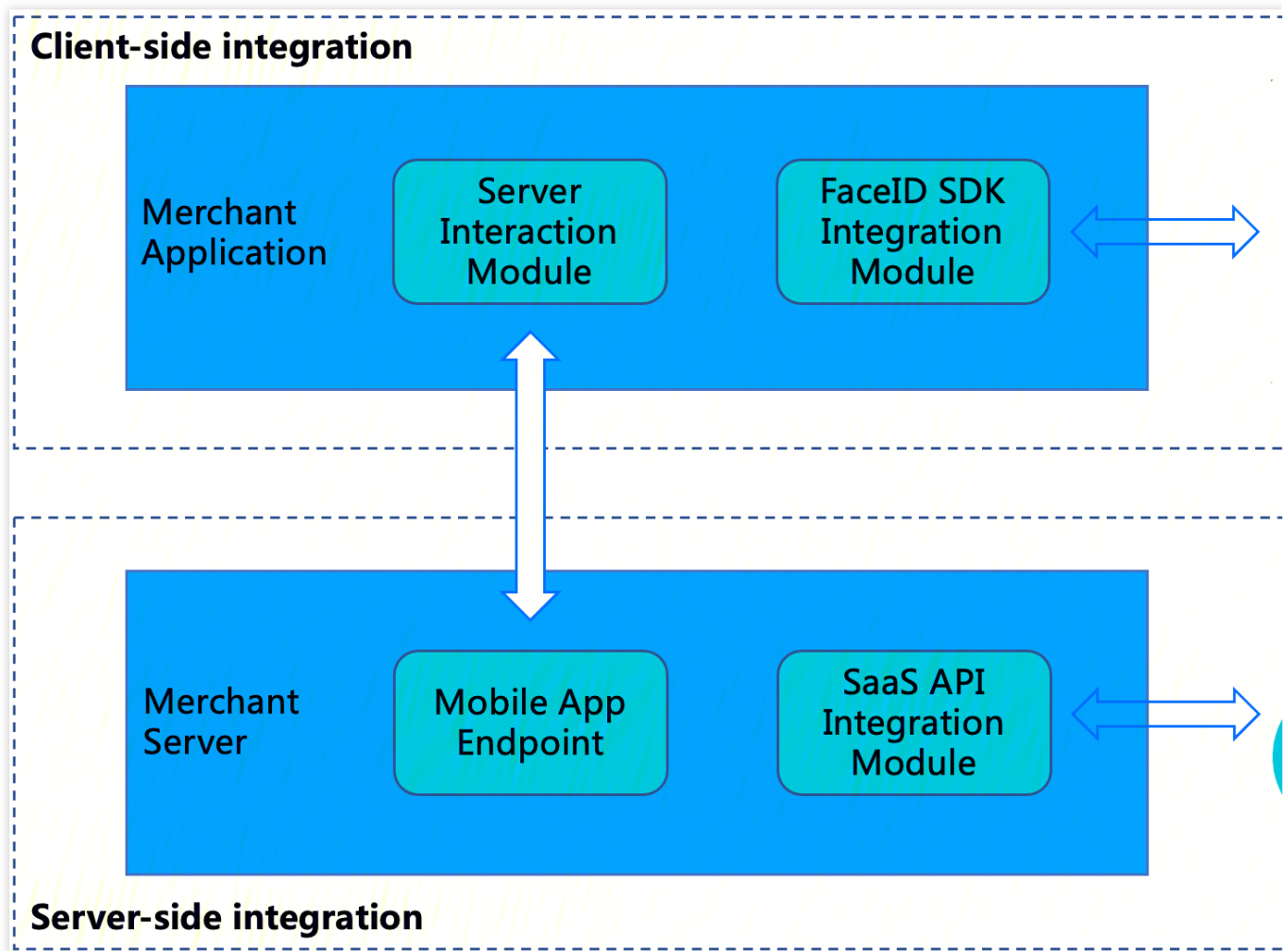
Complete enterprise identity verification. For more information, see [Enterprise Identity Verification Guide](#).

Log in to the [eKYC console](#) and activate the service.

[Contact us](#) to obtain the latest SDK and license.

Overall Architecture Diagram

The following diagram shows the architecture of the liveness detection and face comparison SDK integration.



eKYC SDK integration includes two parts:

Client-side integration: Integrate the eKYC SDK into the customer's terminal service app.

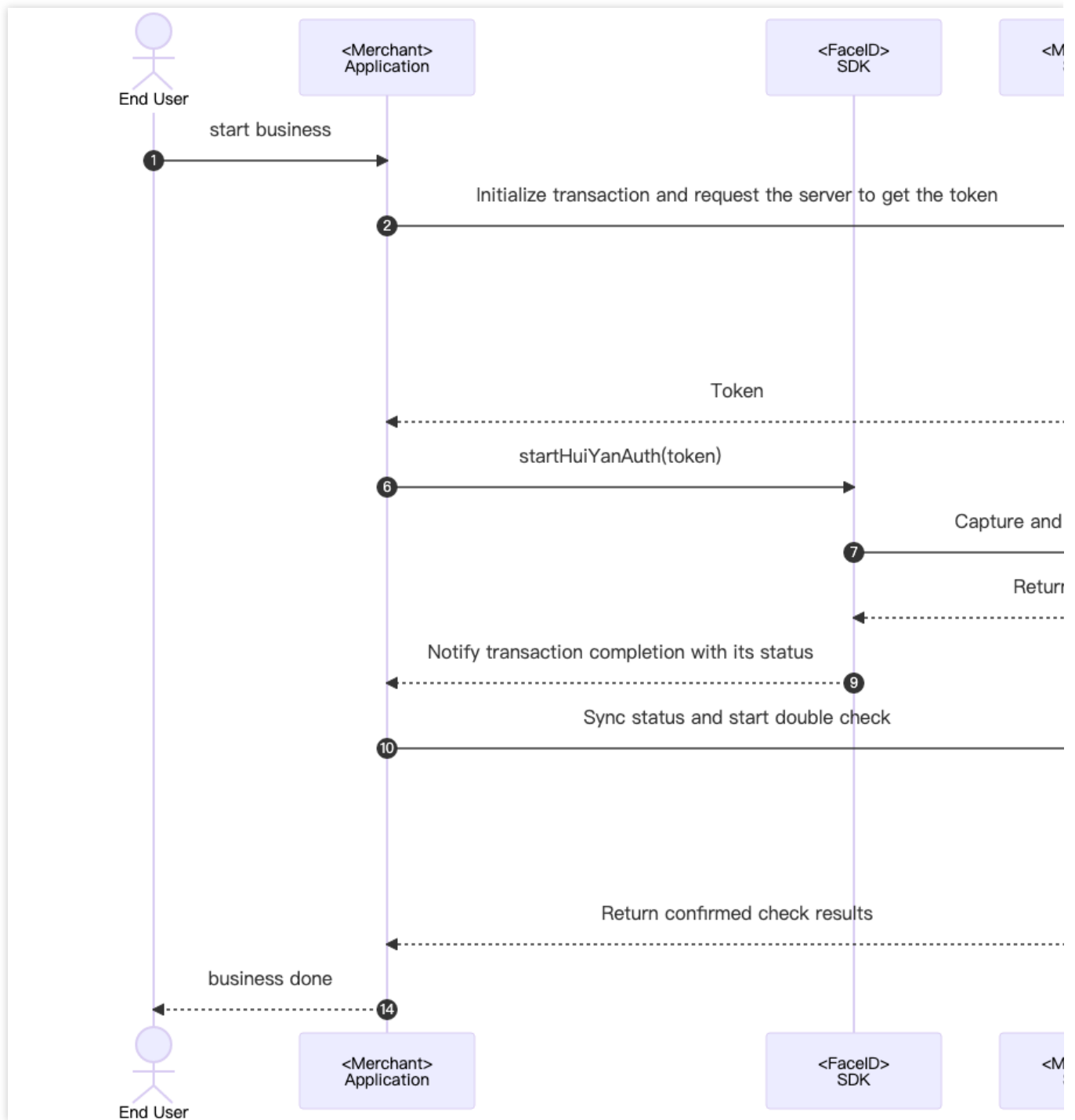
Server-side integration: Expose the endpoint of your (merchant) application to your (merchant) server so that the merchant application can interact with the merchant server and then access the eKYC SaaS API to obtain the `SdkToken`, which is used throughout the liveness detection and face comparison process and to pull the final verification result.

Overall Interaction Process

You only need to pass in the token and start the corresponding eKYC SDK's liveness detection method to complete liveness detection and return the result.

1. TencentCloud API for obtaining the token: [GetFaceldTokenIntl](#)
2. TencentCloud API for pulling the liveness detection result: [GetFaceldResultIntl](#)

The following diagram shows the overall logic of interaction between the SDK, client, and server:



The recommended detailed interaction process is as follows:

1. The customer triggers the merchant application on the terminal to call the liveness verification service scenario.
2. The merchant application sends a request to the merchant server to notify that the liveness detection service token is required for starting liveness verification once.
3. The merchant server passes in relevant parameters to call the TencentCloud API [GetFaceIdTokenIntl](#).
4. After receiving the request for calling [GetFaceIdTokenIntl](#), the FaceID SaaS delivers the service token to the merchant server.
5. The merchant server delivers the obtained service token to the customer's merchant application.

6. The merchant application calls the eKYC SDK's startup API **startHuiYanAuth** to pass in the token and configuration information and starts liveness verification.
7. The eKYC SDK captures and uploads the required user data, including liveness data, to the eKYC SaaS.
8. The eKYC SaaS returns the verification result to the eKYC SDK after completing liveness verification (including the liveness detection and face comparison).
9. The eKYC SDK actively triggers callback to notify the merchant application that the verification is complete and of the verification status.
10. After receiving the callback, the merchant application sends a request to notify the merchant server to obtain the verification result for confirmation.
11. The merchant server actively calls the eKYC SaaS API [GetFaceIdResultIntl](#) to pass in the relevant parameters and service token and obtain the verification result.
12. After receiving the request for calling [GetFaceIdResultIntl](#), the eKYC SaaS returns the verification result to the merchant server.
13. After receiving the verification result, the merchant server delivers the required information to the merchant application.
14. The merchant application displays the final result on the UI to notify the customer of the verification result.

Integration

Server Integration

1. Integration preparations

Before server integration, you need to activate the Tencent Cloud eKYC service and obtain TencentCloud API access key `SecretId` and `SecretKey` by following the instructions in [Getting API Key](#). In addition, you need to follow the instructions in [Connecting to TencentCloud API](#) to import the SDK package with the programming language you are familiar with to your server modules, to ensure that the TencentCloud API can be successfully called and API requests and responses can be properly processed.

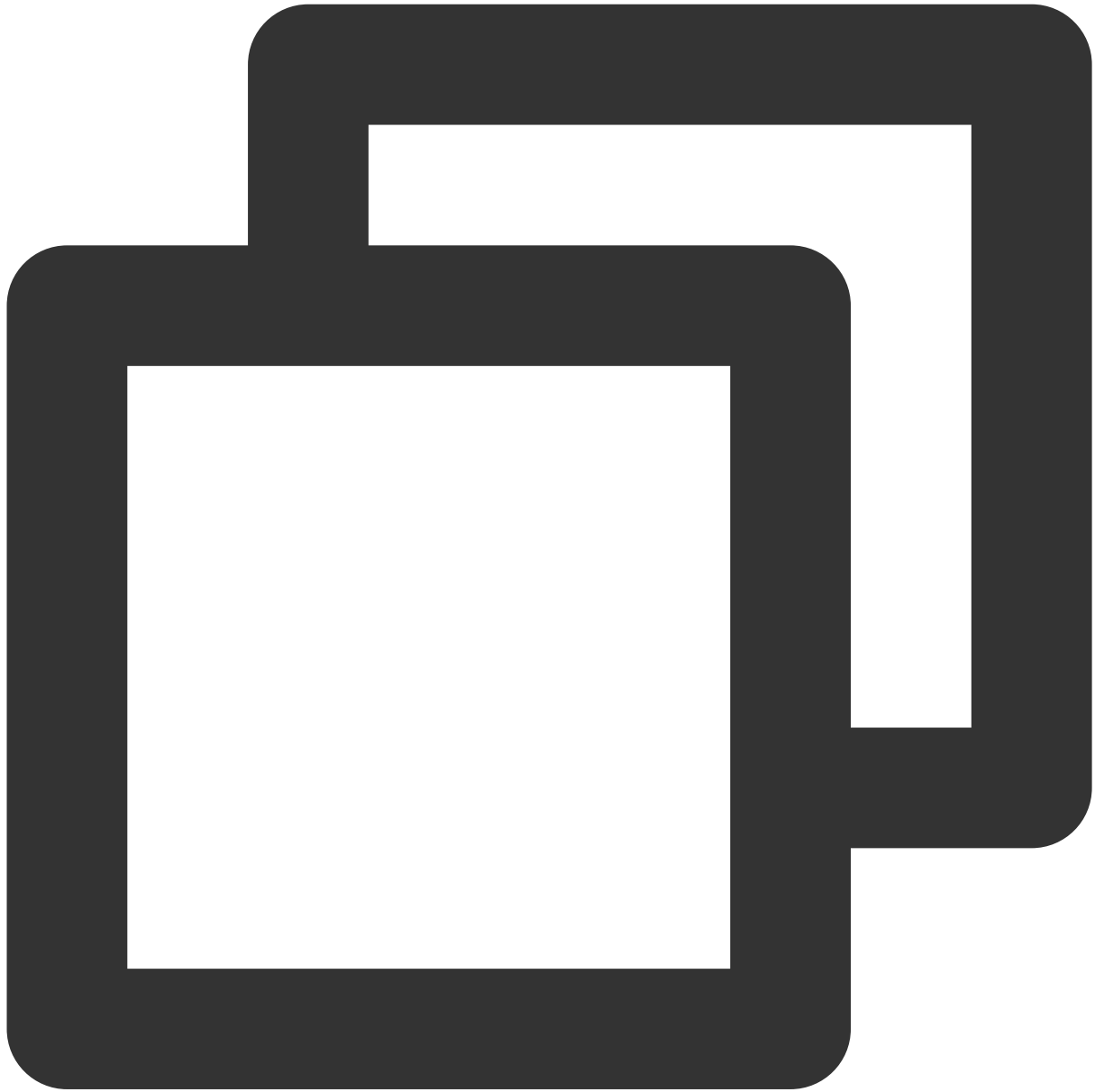
2. Integration process

To ensure that your (merchant) client application interacts with your (merchant) server, the merchant server needs to call the API [GetFaceIdTokenIntl](#) provided by eKYC to obtain `SDKToken`, which is used throughout the liveness detection and face comparison process and used by the API [GetFaceIdResultIntl](#) to obtain the liveness comparison result. The merchant server also needs to provide the corresponding endpoint for the merchant client to call. The following sample code with the Golang language is used as an example to show how to call TencentCloud API on the server and obtain the correct response.

Note: This example only demonstrates the processing logic required for interaction between the merchant server and TencentCloud API service. If necessary, you need to implement your own business logic, for example:

After you obtain the `SDKToken` using the API **GetFaceIdTokenIntl**, you can return other responses required by the client application to the client along with the `SDKToken`.

After you obtain the liveness detection and face comparison result using the API **GetFaceIdResultIntl**, you can save the returned photo with the best frame rate for later business logic.



```
var FaceIdClient *faceid.Client

func init() {
    // Instantiate a client configuration object. You can specify the timeout period.
    prof := profile.NewClientProfile()
    prof.HttpProfile.ReqTimeout = 60
}
```

```
// TODO replace the SecretId and SecretKey string with the API SecretId and Sec
credential := cloud.NewCredential("SecretId", "SecretKey")
var err error
// Instantiate the client object of the requested faceid
FaceIdClient, err = faceid.NewClient(credential, "ap-singapore", prof)
if nil != err {
    log.Fatal("FaceIdClient init error: ", err)
}
}

// GetFaceIdToken get token
func GetFaceIdToken(w http.ResponseWriter, r *http.Request) {
    log.Println("get face id token")
    // Step 1: ... parse parameters
    _ = r.ParseForm()
    var SecureLevel = r.FormValue("SecureLevel")

    // Step 2: instantiate the request object and provide necessary parameters
    request := faceid.NewGetFaceIdTokenIntlRequest()
    request.SecureLevel = &SecureLevel
    // Step 3: call the Tencent Cloud API through FaceIdClient
    response, err := FaceIdClient.GetFaceIdTokenIntl(request)

    // Step 4: process the Tencent Cloud API response and construct the return obje
    if nil != err {
        _, _ = w.Write([]byte("error"))
        return
    }
    SdkToken := response.Response.SdkToken
    apiResp := struct {
        SdkToken *string
    }{SdkToken: SdkToken}
    b, _ := json.Marshal(apiResp)

    // ... more codes are omitted

    //Step 5: return the service response
    _, _ = w.Write(b)
}

// GetFaceIdResult get result
func GetFaceIdResult(w http.ResponseWriter, r *http.Request) {
    // Step 1: ... parse parameters
    _ = r.ParseForm()
    SdkToken := r.FormValue("SdkToken")
    // Step 2: instantiate the request object and provide necessary parameters
    request := faceid.NewGetFaceIdResultIntlRequest()
```

```
request.SdkToken = &SdkToken
// Step 3: call the Tencent Cloud API through FaceIdClient
response, err := FaceIdClient.GetFaceIdResultIntl(request)

// Step 4: process the Tencent Cloud API response and construct the return object
if nil != err {
    _, _ = w.Write([]byte("error"))
    return
}
result := response.Response.Result
apiResp := struct {
    Result *string
}{Result: result}
b, _ := json.Marshal(apiResp)

// ... more codes are omitted

//Step 5: return the service response
_, _ = w.Write(b)
}

func main() {
    // expose endpoints
    http.HandleFunc("/api/v1/get-token", GetFaceIdToken)
    http.HandleFunc("/api/v1/get-result", GetFaceIdResult)
    // listening port
    err := http.ListenAndServe(":8080", nil)
    if nil != err {
        log.Fatal("ListenAndServe error: ", err)
    }
}
```

3. API testing

After you complete the integration, you can test whether the current integration is correct by running the postman or curl command. To be specific, access the API (<http://ip:port/api/v1/get-token>) to check whether `SdkToken` is returned and access the API (<http://ip:port/api/v1/get-result>) to check whether the value of the `Result` field is 0. Through these results, you can determine whether the server integration is successful. For details on responses, see [API introduction](#).

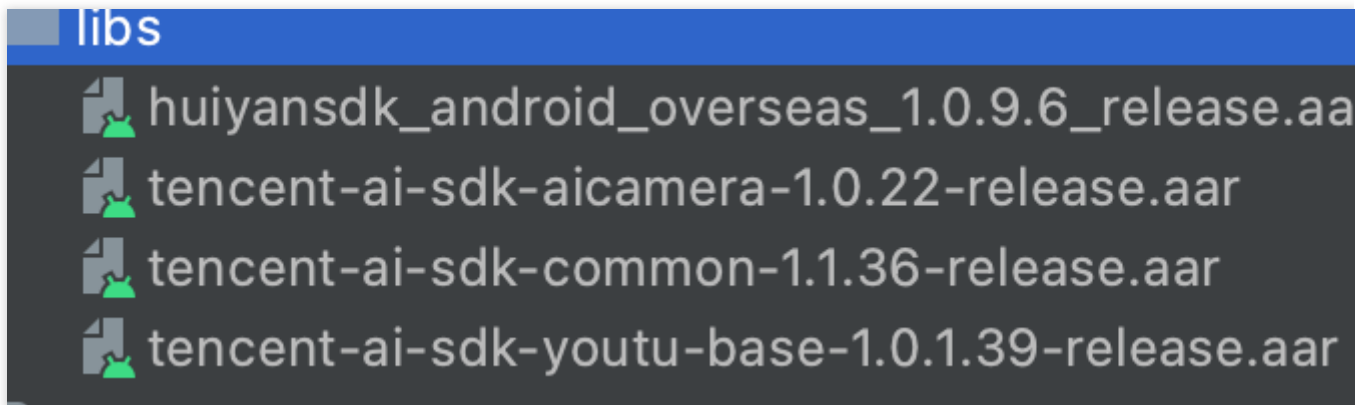
Integration with Android

1. Dependent environment

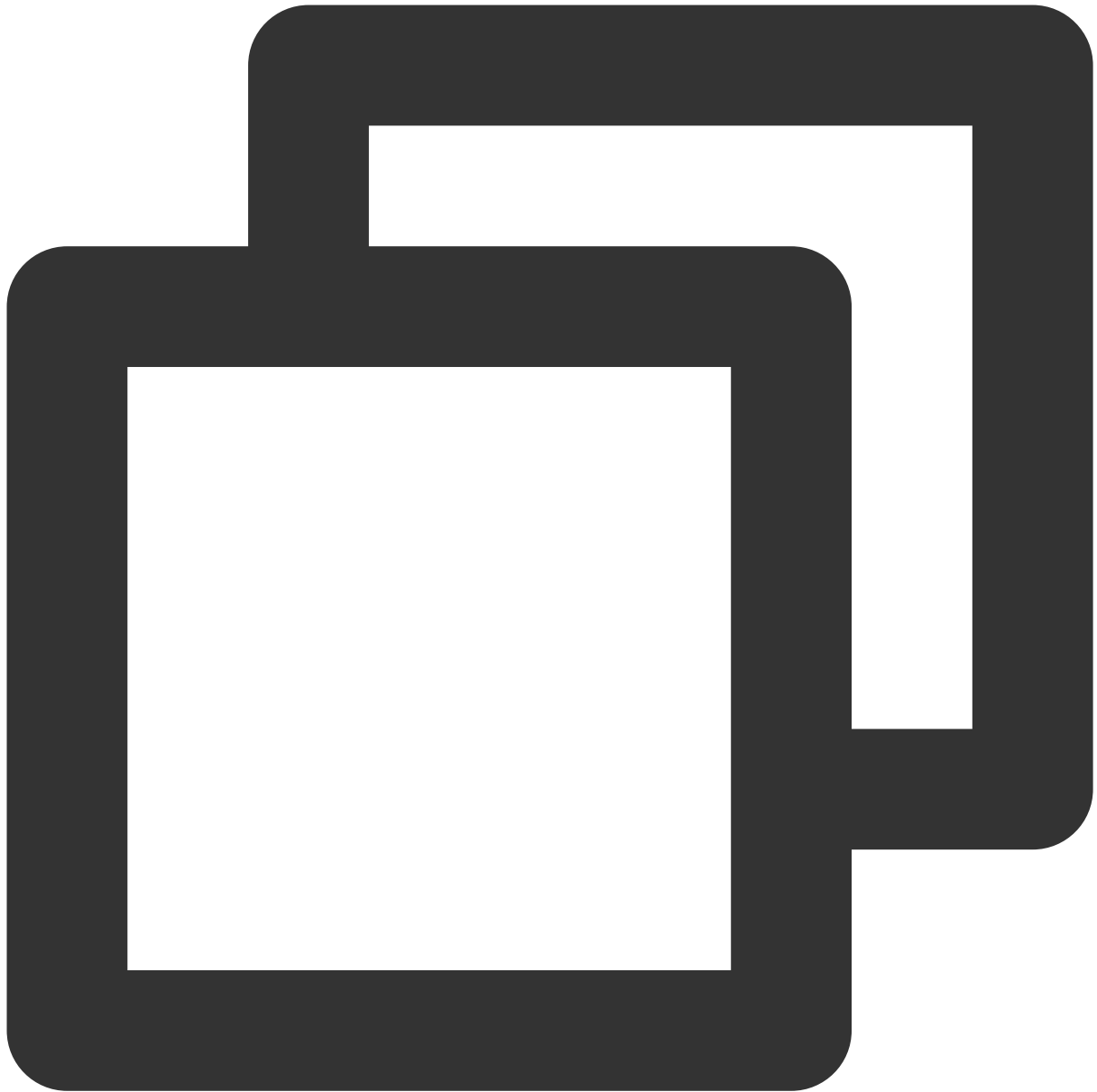
The current FaceID SDK for Android is supported by API 19 (Android 4.4) or later.

2. SDK integration steps

1. Add the files **huiyansdk_android_overseas_1.0.9.6_release.aar**, **tencent-ai-sdk-youtu-base-1.0.1.39-release.aar**, **tencent-ai-sdk-common-1.1.36-release.aar**, and **tencent-ai-sdk-aicamera-1.0.22-release.aar** (the specific version numbers of the files downloaded from the official website shall prevail) to the **libs** directory of your project.



2. Configure **build.gradle** in your project as follows:



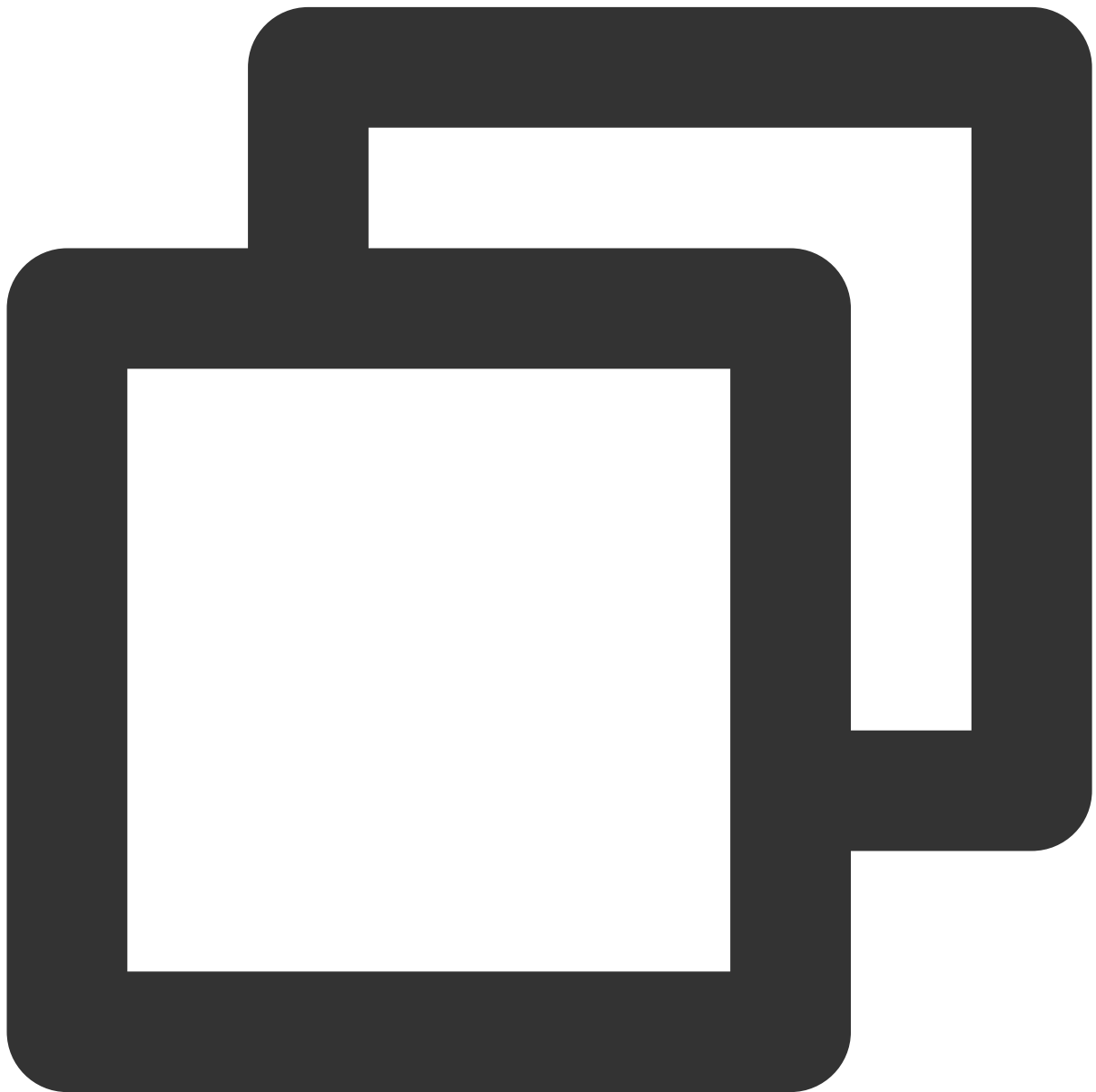
```
// Set .so architecture filtering in NDK (using armeabi-v7a as an example)
ndk {
    abiFilters 'armeabi-v7a'
}

dependencies {
    // Import the FaceID SDK
    implementation files("libs/huiyansdk_android_overseas_1.0.9.5_release.aar")
    // FaceID general algorithm SDK
    implementation files("libs/tencent-ai-sdk-youtu-base-1.0.1.32-release.aar")
    // Common capability components
```

```
implementation files("libs/tencent-ai-sdk-common-1.1.27-release.aar")
implementation files("libs/tencent-ai-sdk-aicamera-1.0.18-release.aar")

// Third-Party libraries that the FaceID SDK depends on
// gson
implementation 'com.google.code.gson:gson:2.8.5'
}
```

3. Make the necessary permission declaration in the `AndroidManifest.xml` file.



```
<!-- Camera permission -->
```

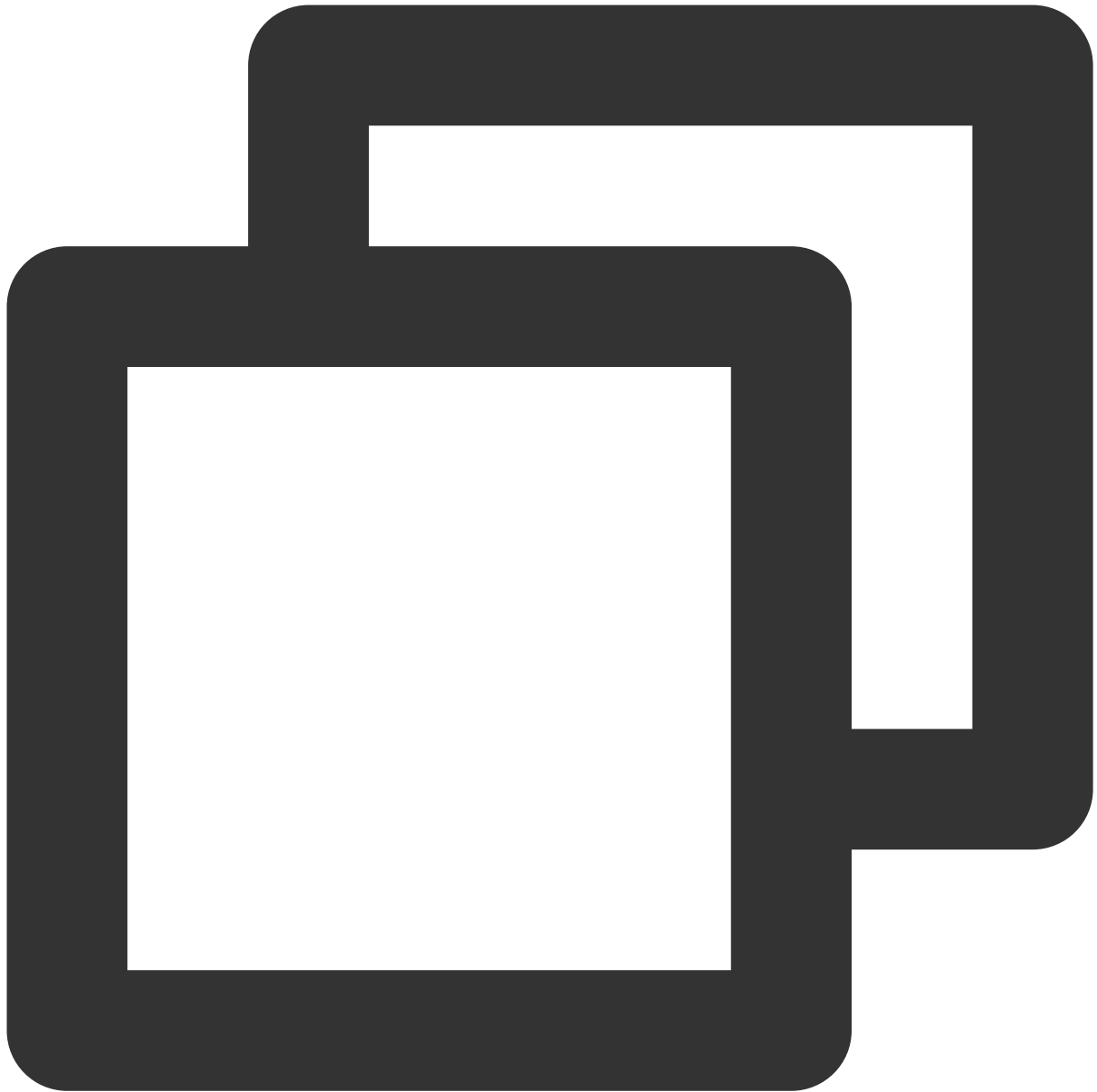
```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />
<uses-feature android:name="android.hardware.camera.autofocus" />

<!-- Permissions required by the SDK -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- Optional permissions for the SDK -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

If your app needs to be compatible with Android 6.0 or later, in addition to declaring the above permissions in the `AndroidManifest.xml` file, you need to add the code **Dynamically apply for permissions**.

3. API initialization

This API is called during app initialization, which is mainly used to perform some initialization operations for the SDK. We recommend you call this API in `Application`.



```
@Override
public void onCreate() {
    super.onCreate();
    instance = this;
    // Initialize the SDK during app initialization
    HuiYanOsApi.init(getApp());
}
```

4. Start the liveness verification API



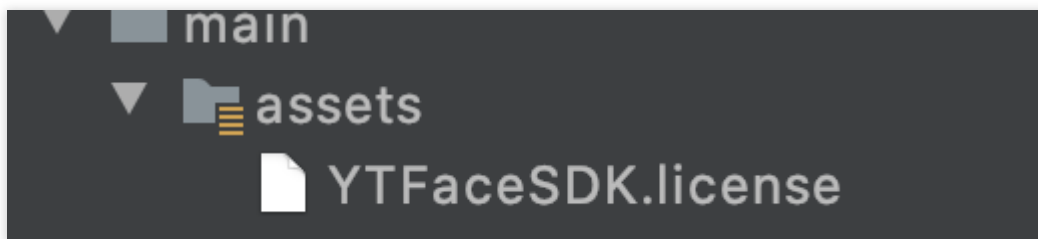
```
// HuiYanOs parameters
HuiYanOsConfig huiYanOsConfig = new HuiYanOsConfig();
// The license file is placed in `assets`.
huiYanOsConfig.setAuthLicense("YTFaceSDK.license");
if (compatCheckBox.isChecked()) {
    huiYanOsConfig.setPageColorStyle(PageColorStyle.Dark);
}
// Whether to return the best frame
if (needBestImageCB.isChecked()) {
    huiYanOsConfig.setNeedBestImage(true);
}
```

```
// Start liveness verification. `currentToken` is the token distributed by the back
HuiYanOsApi.startHuiYanAuth(currentToken, huiYanOsConfig, new HuiYanOsAuthCallBack(
    @Override
    public void onSuccess(HuiYanOsAuthResult authResult) {
        showToast("Liveness verification passed.");
        if (!TextUtils.isEmpty(authResult.getBestImage())) {
            CommonUtils.decryptBestImgBase64(authResult.getBestImage(), false);
        }
    }

    @Override
    public void onFail(int errorCode, String errorMsg, String token) {
        String msg = "Liveness verification failed " + "code: " + errorCode + " msg"
        Log.e(TAG, "onFail" + msg);
        showToast(msg);
    }
});
```

[HuiYanOsAuthResult](#) is the returned result of successful liveness verification. The final liveness verification result can be obtained by accessing [GetFaceldResultIntl](#) through the token.

Note: You need to contact the customer service to apply for the "YTFaceSDK.license" file, and then place the license file in the `Assets Folder` .



5. Release SDK resources

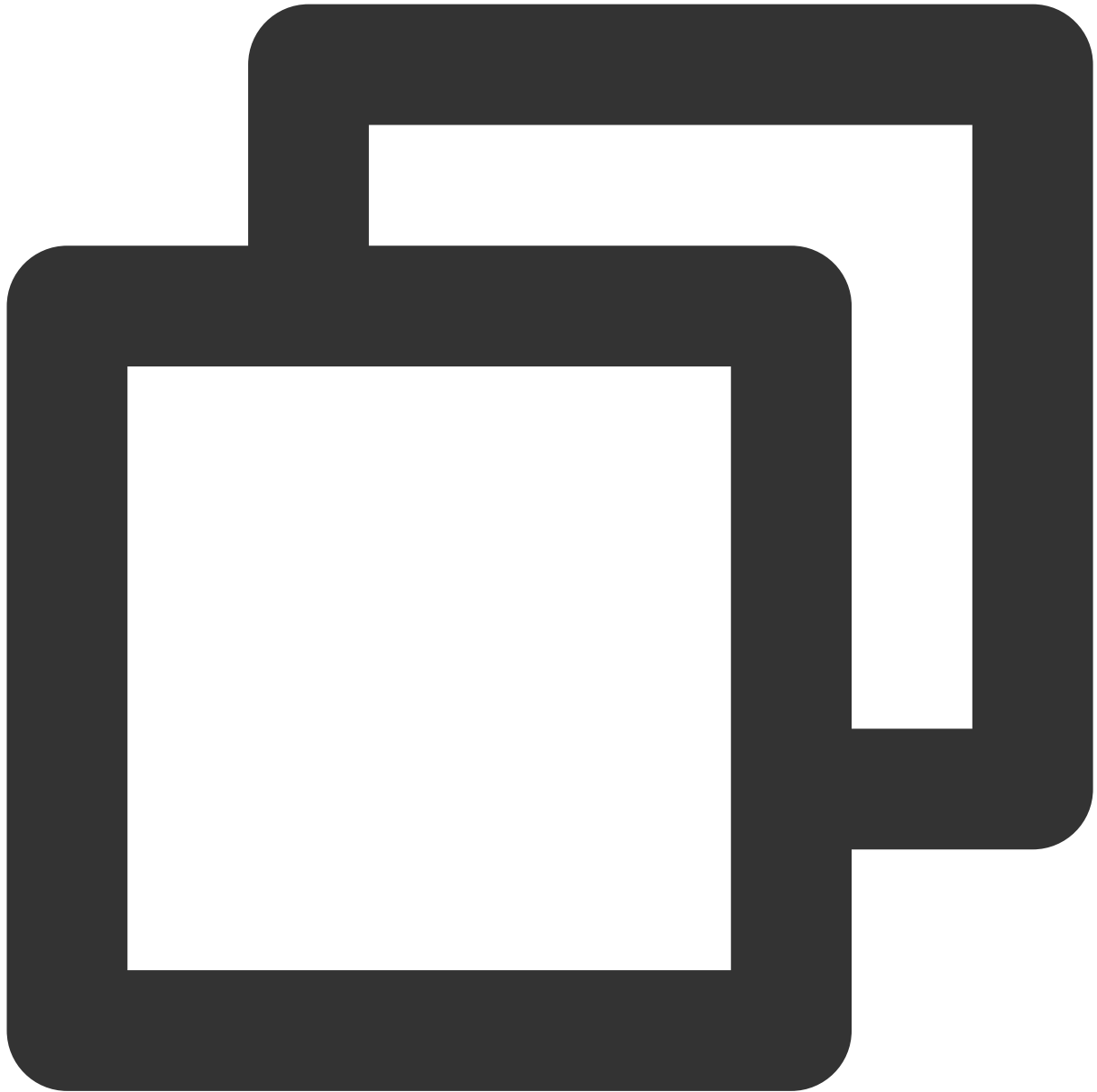
Before your app exits, you can call the API to release SDK resources.



```
@Override
protected void onDestroy() {
    super.onDestroy();
    // Release the resources upon exit
    HuiYanOsApi.release();
}
```

6. Configure obfuscation rules

If the obfuscation feature is enabled for your app, add the following to your obfuscation file to ensure the normal running of the SDK:



```
# The following FaceID SDK obfuscation rules should be added:
-keep class com.tencent.could.huiyansdk.** {*; }
-keep class com.tencent.could.aicamare.** {*; }
-keep class com.tencent.could.component.** {*; }
-keep class com.tencent.youtu.** {*; }
-keep class com.tenpay.utils.SMUtils {*; }
```

Integration with iOS

1. Dependent environment

1. Development environment: Xcode 11.0 or later
2. The eKYC SDK for iOS is only supported by iOS 9.0 or later.

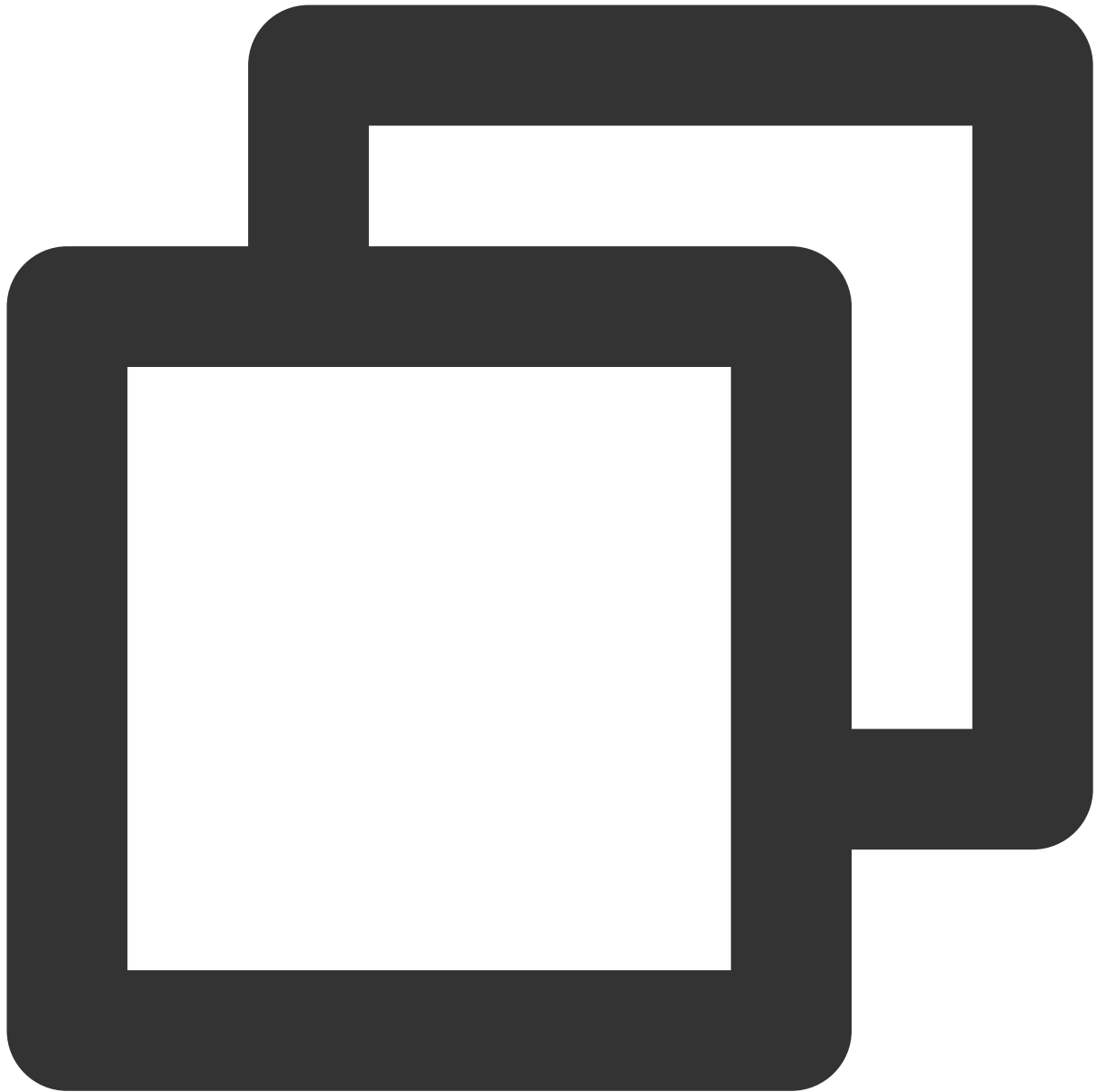
2. SDK integration steps

Manual integration

1. Import libraries and files.

Click `Link Binary With Libraries` to add frameworks.

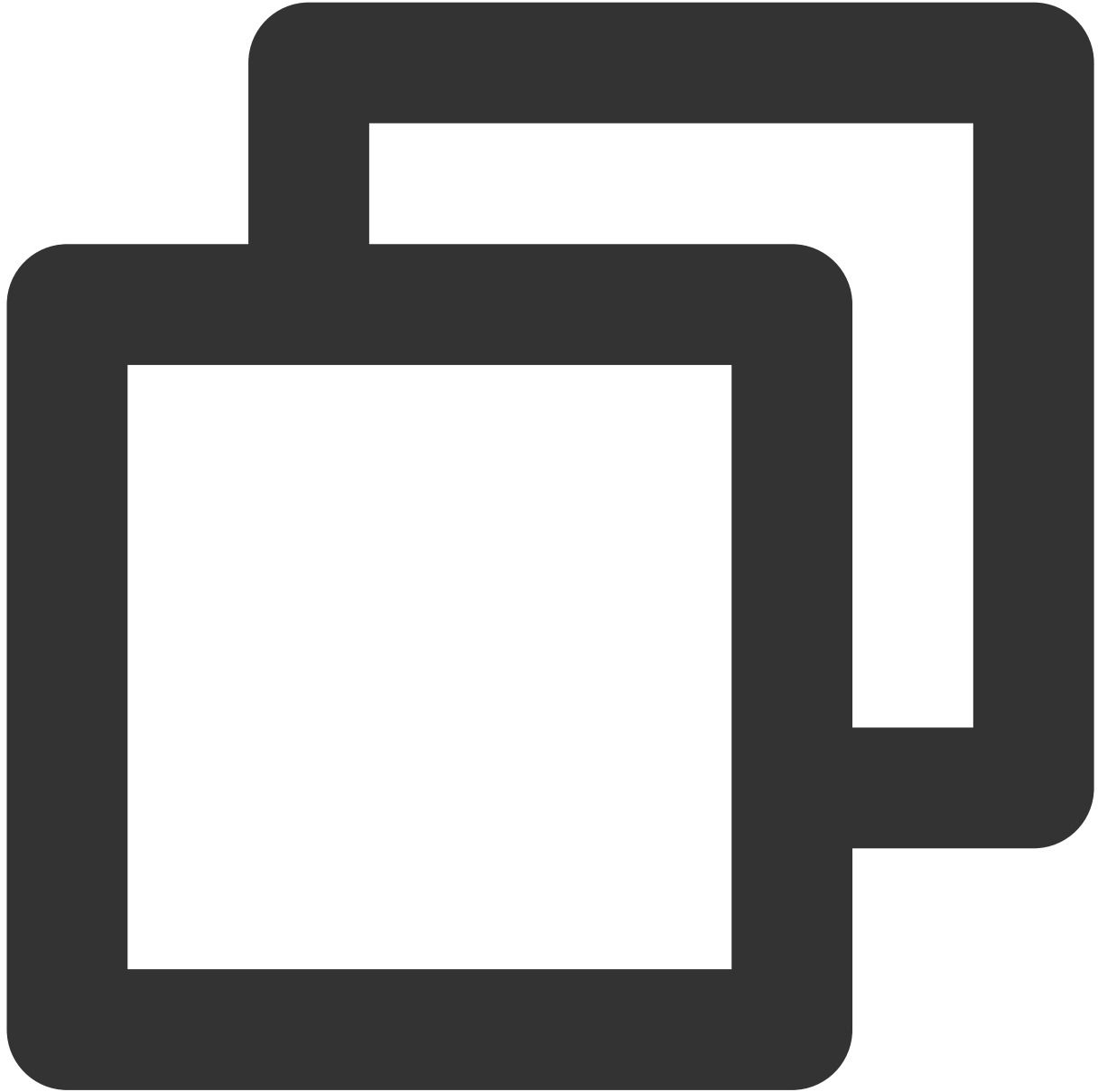
2. The SDK depends on the following libraries:



```
├─HuiYanSDK.framework
├─YtSDKKitSilentLiveness.framework
├─YtSDKKitReflectLiveness.framework
├─YtSDKKitActionLiveness.framework
├─YtSDKKitFramework.framework
├─tnnliveness.framework
├─YTFaceAlignmentTinyLiveness.framework
├─YTFaceTrackerLiveness.framework
├─YTFaceDetectorLiveness.framework
├─YTPoseDetector.framework
├─YTCommonLiveness.framework
```

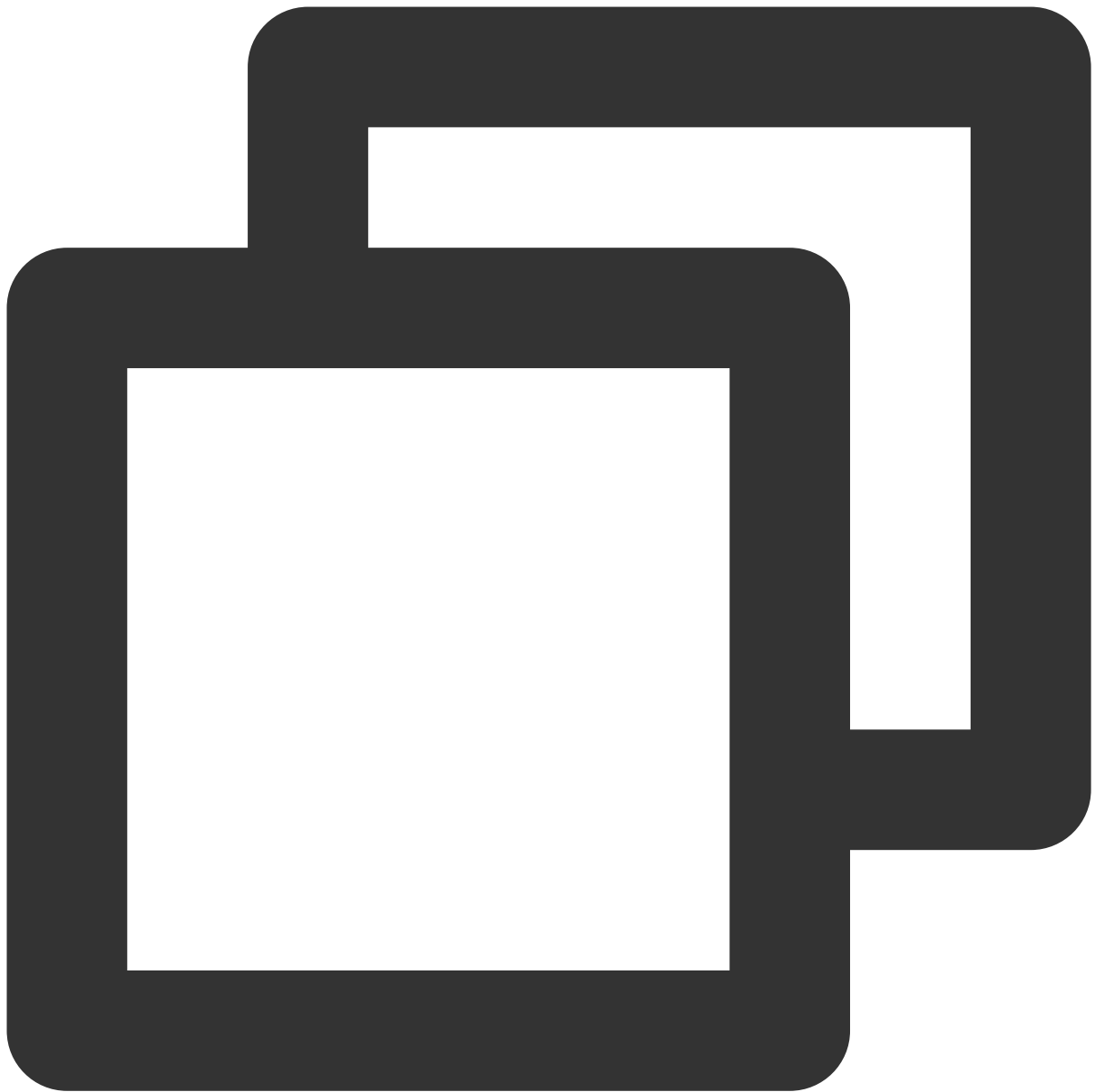
```
└─YTFaceLiveReflect.framework
```

3. Click `Link Binary With Libraries` to add system frameworks.



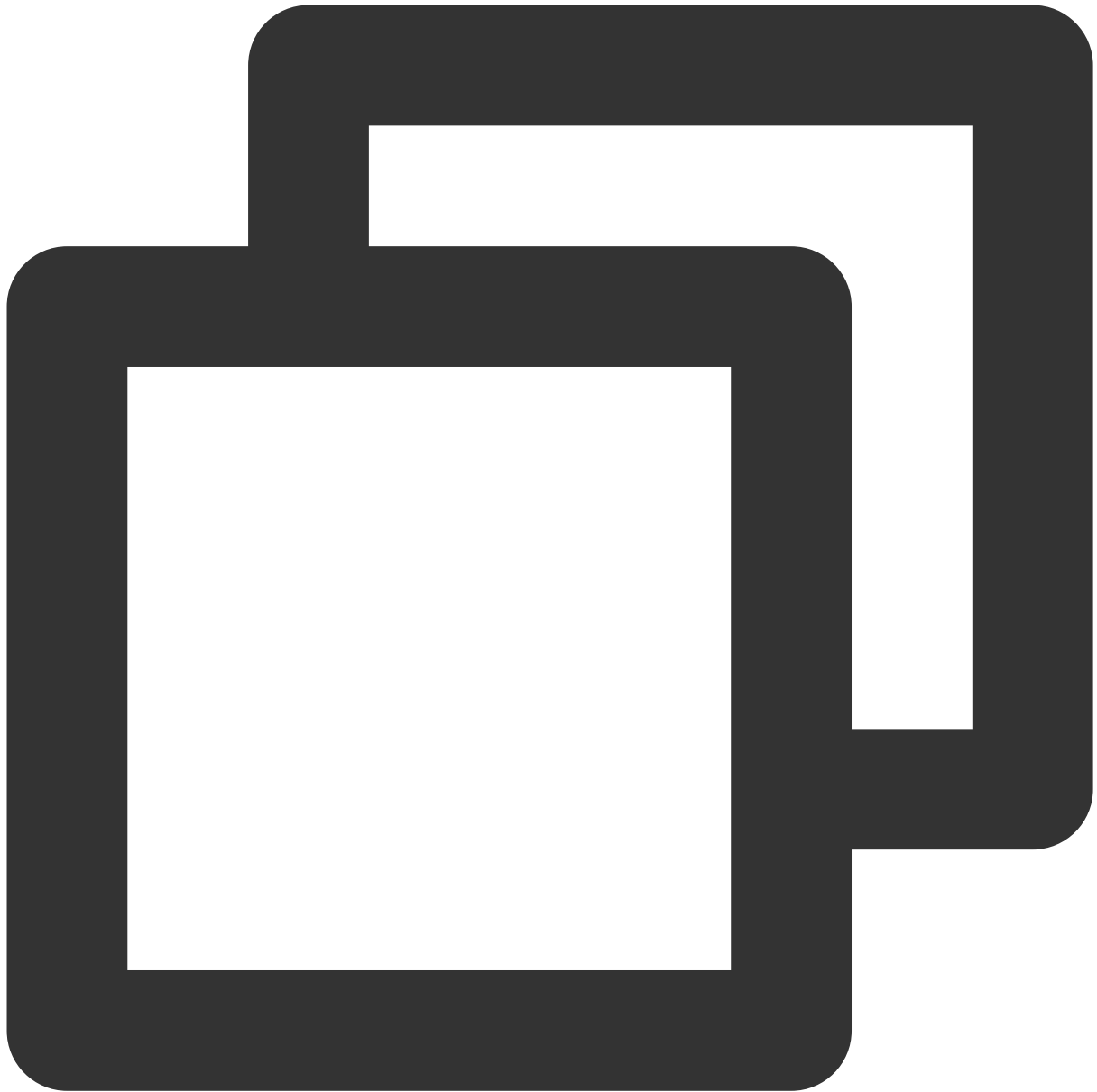
```
└─ AVFoundation.framework  
└─ libc++.tbd  
└─ Accelerate.framework
```

4. Import the model in `Copy Bundle Resources` .



└─ face-tracker-v001.bundle

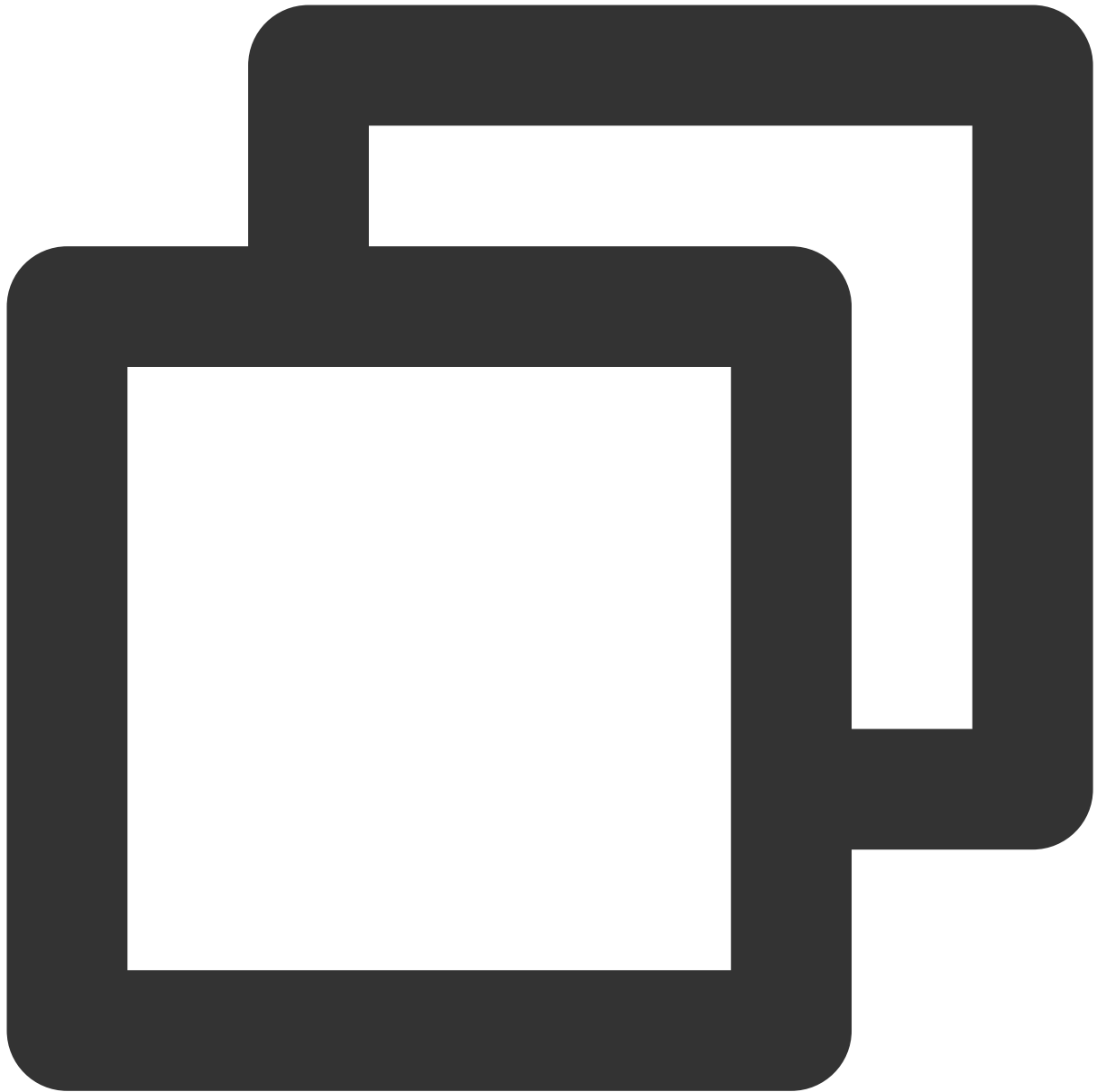
5. Import the resource file in `Copy Bundle Resources` .



└─ HuiYanSDKUI.bundle

Integration by using pod

1. Copy the `CloudHuiYanSDK_FW` folder to the directory at the same level as that of the integration project `Podfile`.
2. Set the following in `Podfile` :



```
target 'HuiYanAuthDemo' do
  use_frameworks!
  pod 'CloudHuiYanSDK_FW', :path => './CloudHuiYanSDK_FW'
end
```

3. Run `pod install` .

Note:

For the file levels and specific settings, see the demo.

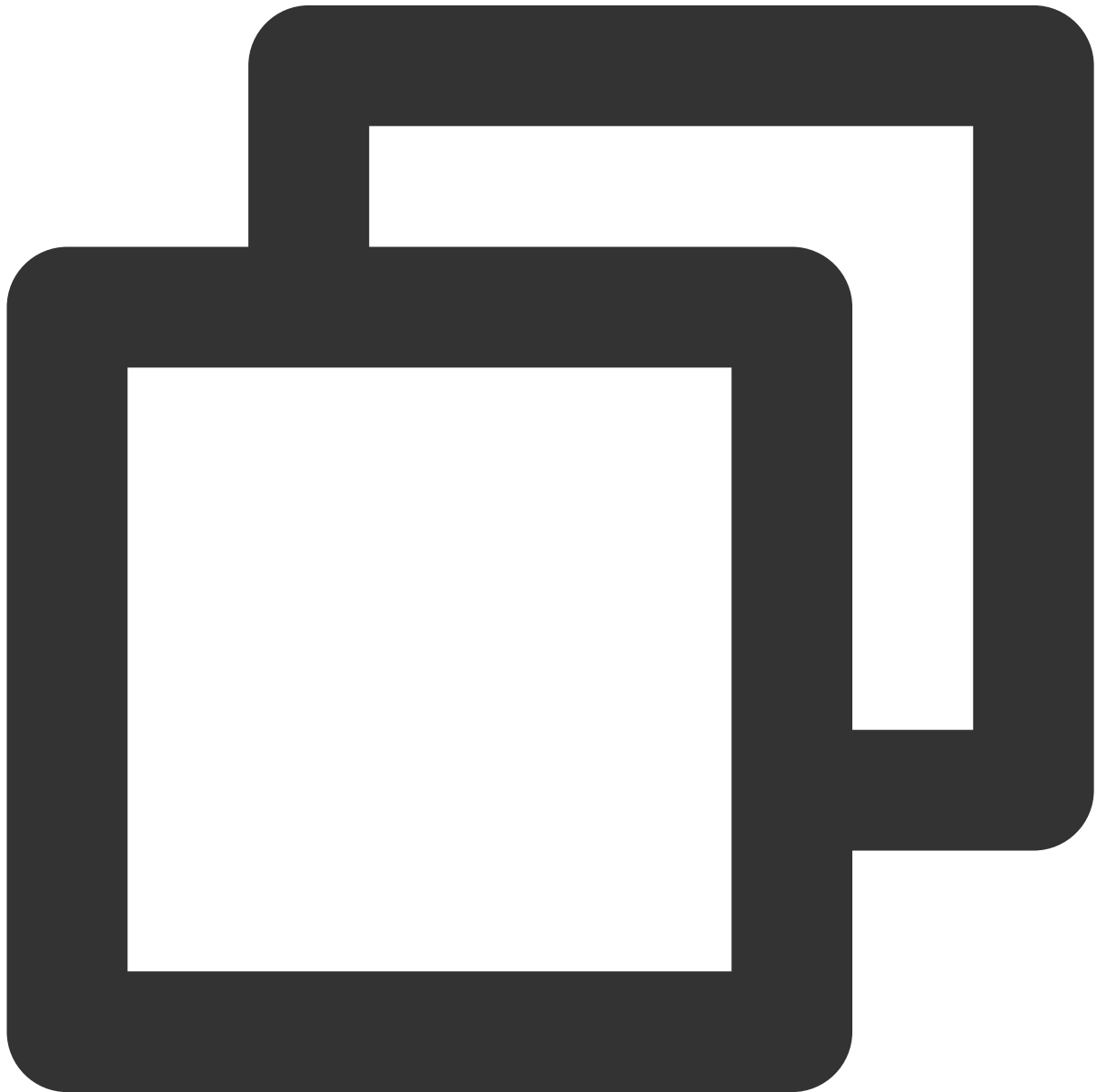
[iOS demo](#)

Build Phases settings

1. Click `Other Linker Flags` to add **-ObjC**.
2. Integrate `ViewController.m` and set the extension to `.mm` (for a Swift project, add the system library `libc++.tbd`).

Permission settings

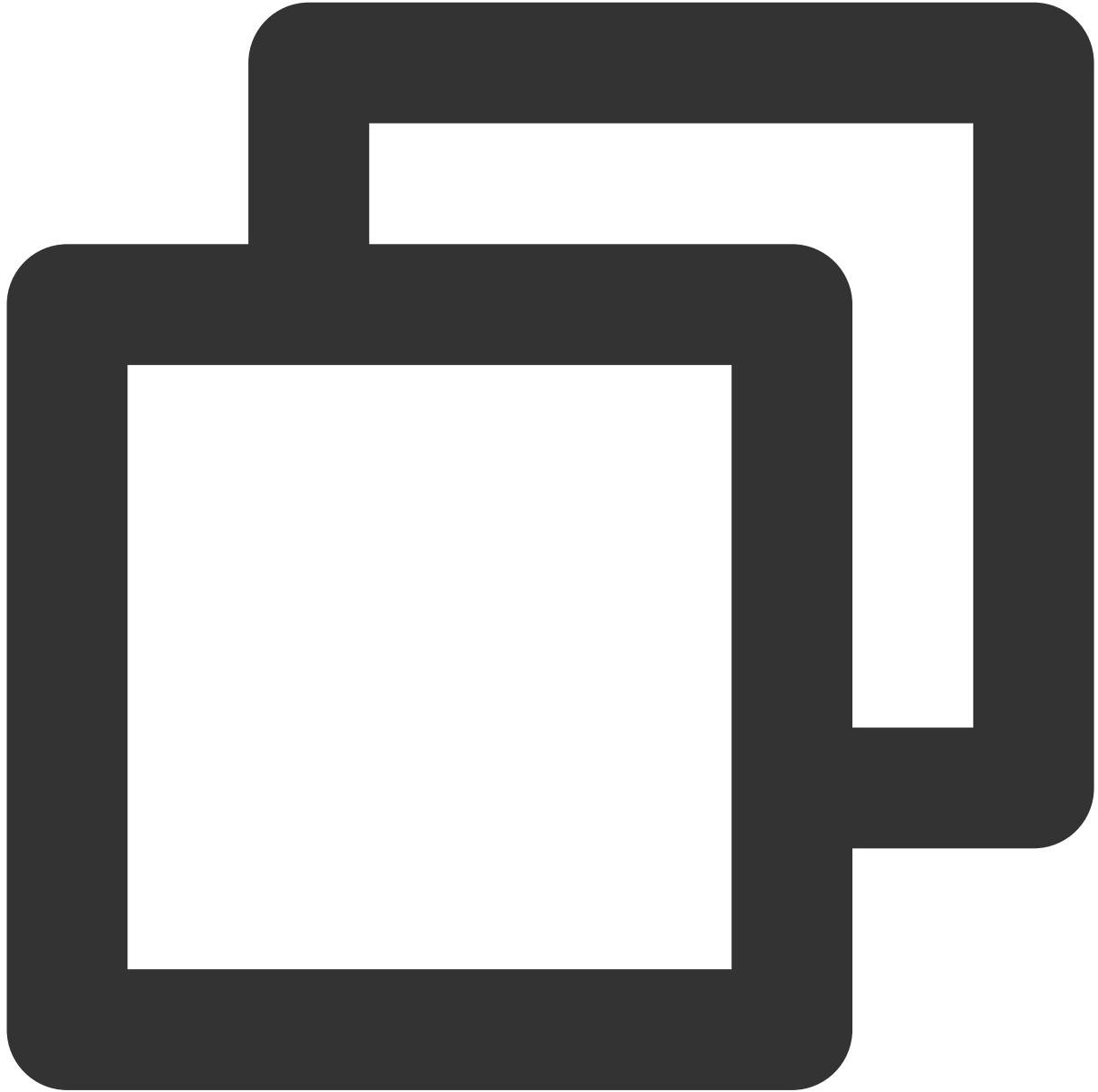
As the SDK requires a mobile network and camera permission, include the following key-value pair in `info.plist` of the main project to add the corresponding permission declaration.



```
<key>Privacy - Camera Usage Description</key>
```

```
<string>FaceID requires you to grant the camera permission for face recognition.</s
```

3. Start the liveness detection API



```
#import <HuiYanSDK/HuiYanOsApi.h>
#import <HuiYanSDK/HuiYanOSKit.h>

// Get the token
NSString *faceToken = self.tokenTextField.text;
// Configure the SDK
HuiYanOsConfig *config = [[HuiYanOsConfig alloc] init];
```

```
// Set the license
config.authLicense = [[NSBundle mainBundle] pathForResource:@"xxx.lic" ofType:@
// Timeout configuration for the preparation stage
config.prepareTimeoutMs = 20000;
// Timeout configuration for the detection stage
config.authTimeOutMs = 20000;
config.isDeleteVideoCache = YES;
config.languageType = EN;
//      config.userLanguageFileName = @"ko";
//      config.userLanguageBundleName = @"UseLanguageBundle";
config.iShowTipsPage = YES;
config.isGetBestImg = YES;

[[HuiYanOSKit sharedInstance] startHuiYaneKYC:faceToken withConfig:config
                                witSuccCallback:^(HuiYanOsAuthResult * _Nonnull a
    NSString *bestImg = authResult.bestImage;
    NSString *token = authResult.faceToken;

} withFailCallback:^(int errCode, NSString * _Nonnull errMsg, id _Nullable res
    NSString *showMsg = [NSString stringWithFormat:@"err:%d:%@",errCode,errMsg]
    NSLog(@"err:%@", showMsg);
}];
```

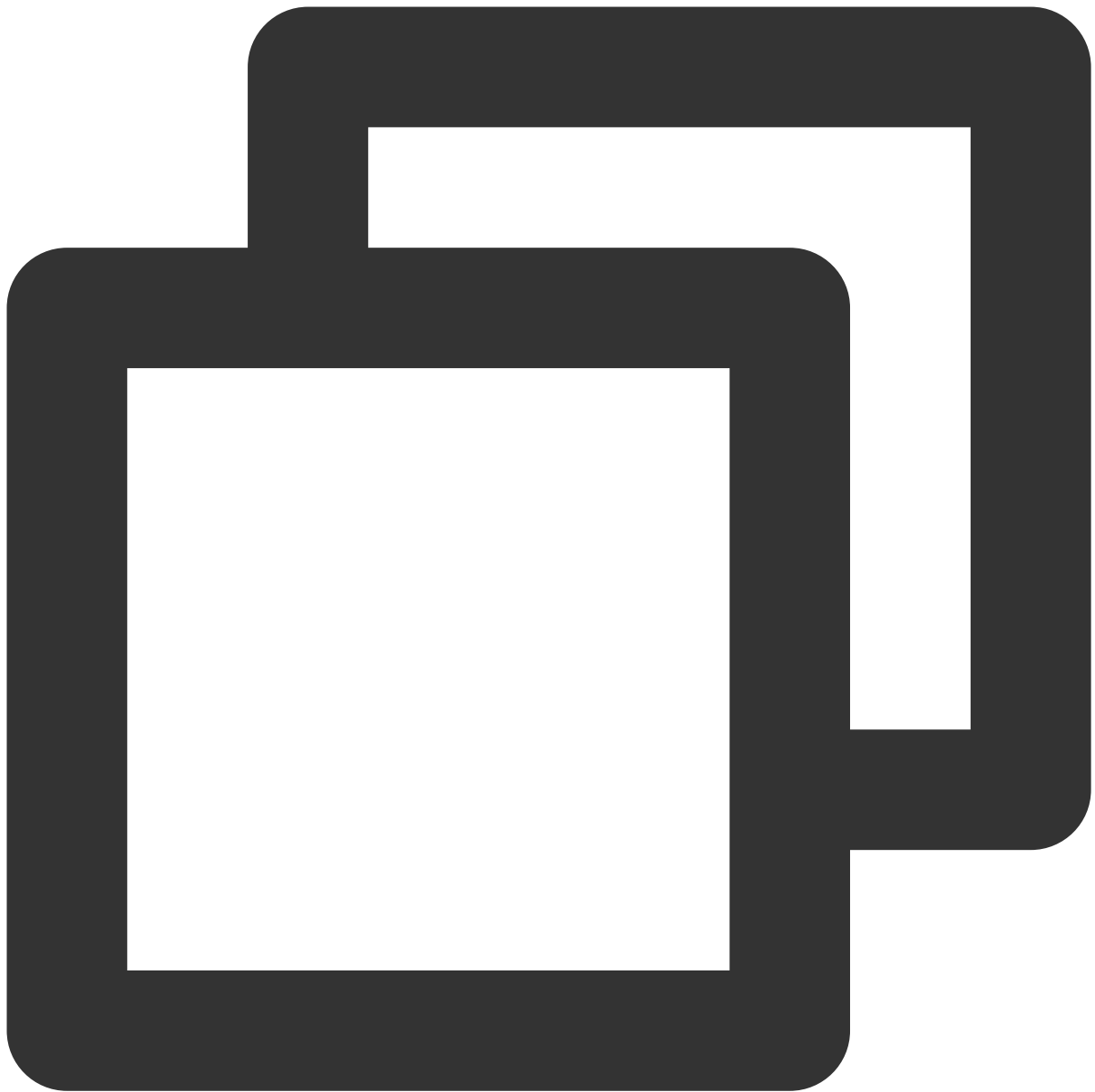
[HuiYanOsAuthResult](#) is the returned result of successful liveness verification. The final liveness verification result can be obtained by accessing [GetFaceldResultIntf](#) through the token.

Note:

Currently, you need to contact the customer service to apply for the "**xxx.lic**" file actively.

4. Release SDK resources

Before your app exits, you can call the API to release SDK resources.



```
// Release the resources before exit
- (void)dealloc {
    [HuiYanOsApi release];
}
//[iOS demo] (https://github.com/TencentCloud/huiyan-faceid-demo/tree/main/faceid-iOS)
[Android demo] (https://github.com/TencentCloud/huiyan-faceid-demo/tree/main/id-verify)
```

[iOS demo](#)[Android demo](#)

SDK API Description

Android API Description

Last updated : 2023-06-13 11:24:06

API description

The eKYC SDK mainly involves the following classes: `HuiYanOsApi` (API class), `HuiYanOsConfig` (parameter configuration class), and `HuiYanOsAuthCallBack` and `HuiYanAuthEventCallBack` (result callback classes).

HuiYanOsApi

API	Feature Description
<code>init()</code>	Initializes the service.
<code>release()</code>	Releases resources.
<code>setAuthEventCallBack()</code>	Sets the key actions in the liveness detection and face comparison process.
<code>startHuiYanAuth()</code>	Performs liveness detection and face comparison. This API can be called to complete the entire process.

`init()`

Feature description:

This API is used to initialize the eKYC SDK.



```
public static void init(Context context)
```

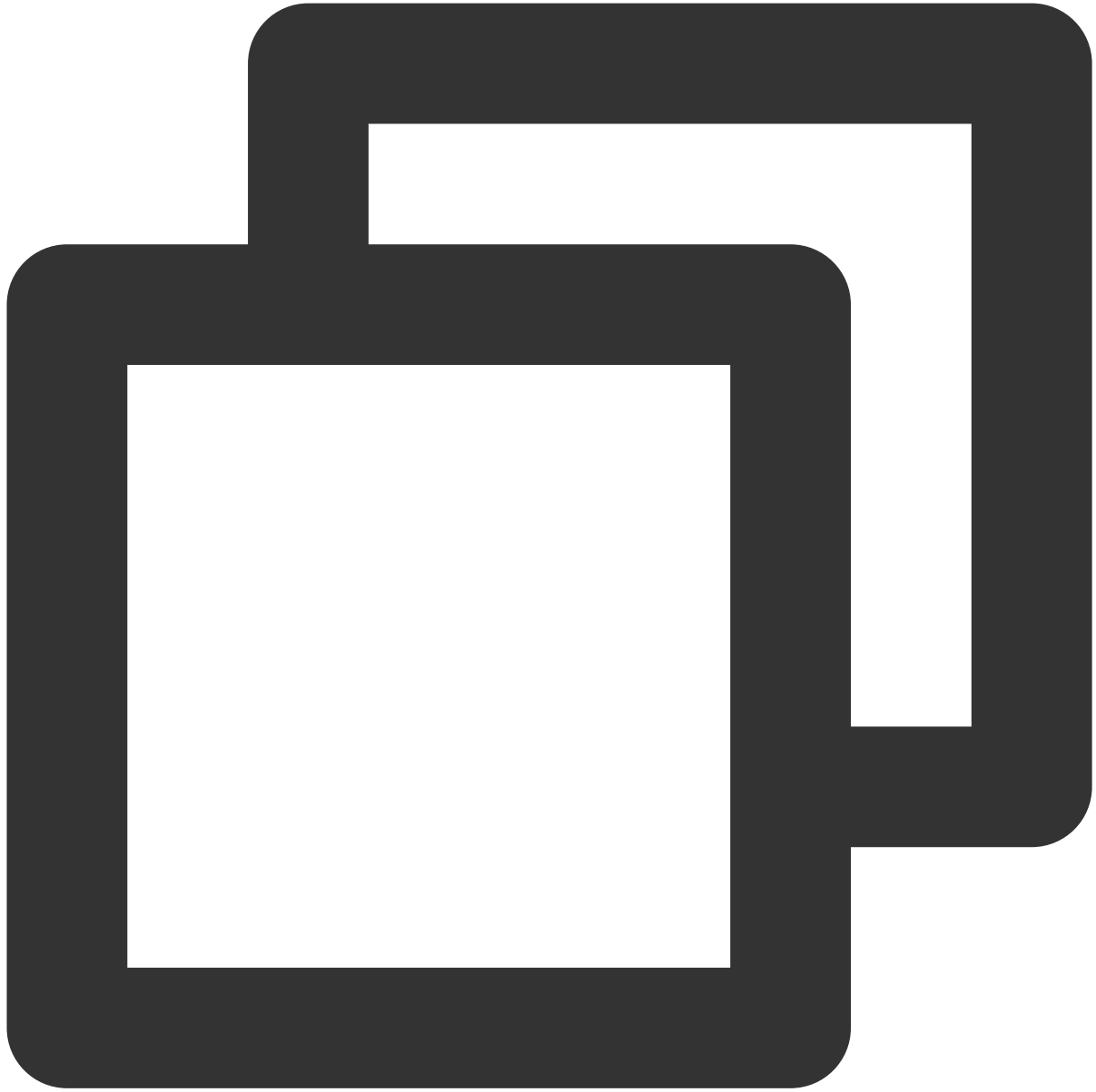
Input parameters:

Type	Parameter	Description
Context	context	App context information

release()

Feature description:

This API is used to release eKYC SDK resources.



```
public static void release()
```

setAuthEventCallBack()

Feature description:

This API is used to set callback for the key actions in the liveness detection and face comparison process.



```
public static void setAuthEventCallBack(HuiYanAuthEventCallBack authEventCallBack)
```

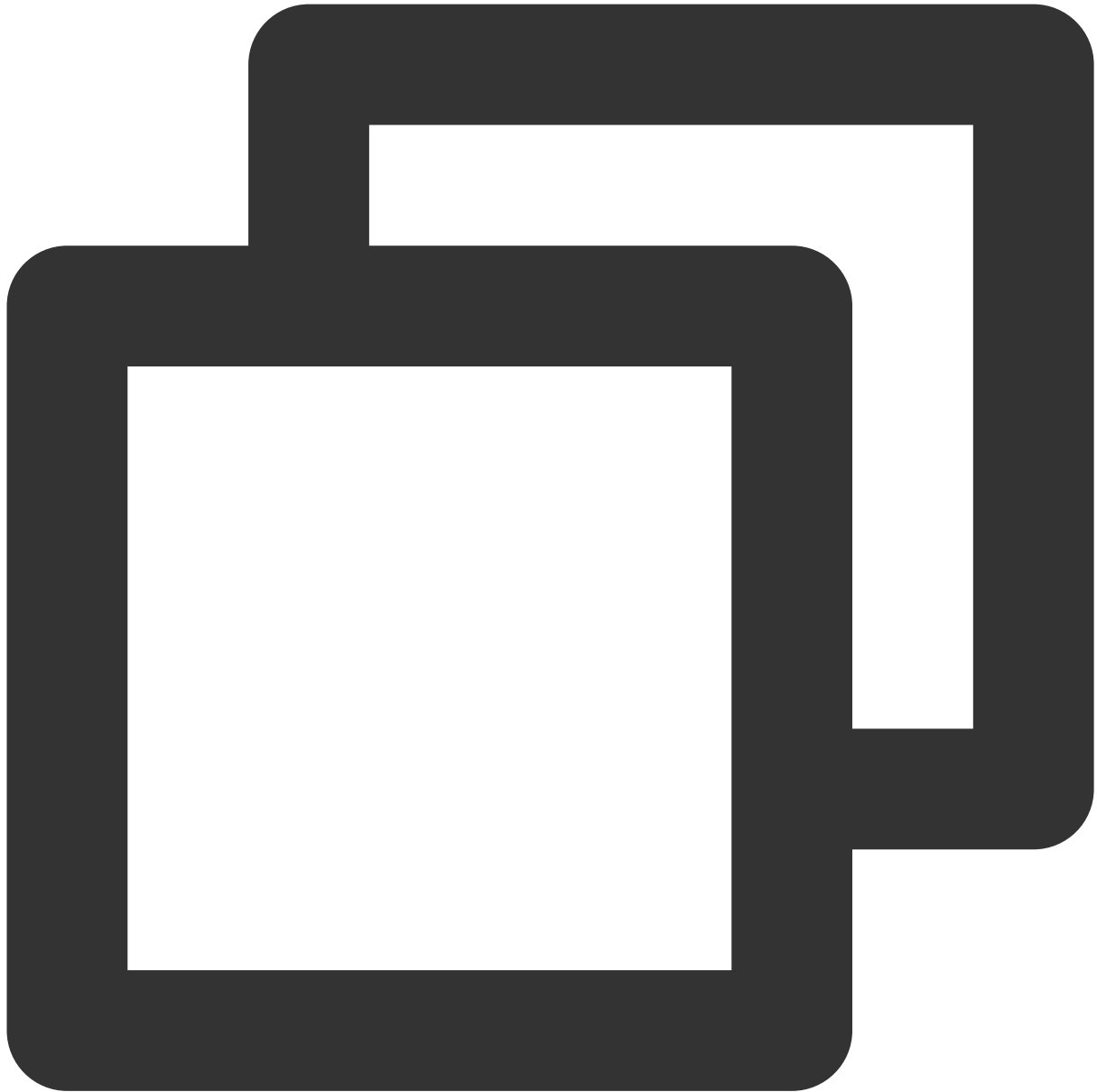
Input parameters:

Type	Parameter	Description
HuiYanAuthEventCallBack	huiYanAuthEventCallBack	Callback for key liveness detection and face comparison actions

startHuiYanAuth()

Feature description:

This API is used for liveness detection and face comparison. It can be called to complete the entire process.



```
public static void startHuiYanAuth(final String startToken, final HuiYanOsConfig st
```

Input parameters:

Type	Parameter	Description
String	startToken	Business token requested from the server for starting liveness detection and face comparison

HuiYanOsConfig	startConfig	Configuration parameter
HuiYanOsAuthCallBack	authCallBack	liveness detection and face comparison result

HuiYanOsConfig

`HuiYanOsConfig` is the configuration entity class during eKYC SDK startup, which mainly contains the following attributes:

Type	Name	Description	Default Value
<code>PageColorStyle</code>	pageColorStyle	Color pattern used for liveness detection and face comparison	<code>PageColorStyle.Light</code>
String	authLicense	Name of the license file applied for by client for liveness detection and face comparison authorization	Empty
long	authTimeOutMs	Liveness detection and face comparison timeout period	10000 ms (10s)
boolean	isDeleteVideoCache	Whether to delete the local cache of the liveness detection and face comparison video	true
boolean	isShowGuidePage	Whether to open the guide page for liveness detection and face comparison	true
boolean	isNeedBestImage	Whether to return the best frame	false
LanguageStyle	languageStyle	Language type	Auto
String	languageCode	Language code (see Language codes for Android), which is used together with <code>languageStyle</code>	Empty
String	backUpIPs	List of backup IPs	Empty
String	backUpHost	Backup domain	Empty
AuthUiConfig	authUiConfig	UI configuration items	Empty

PageColorStyle

This is the enumeration class of default color patterns on the default liveness detection and face comparison UI. Currently, two color patterns are supported: light and dark.

PageColorStyle Value	Description
----------------------	-------------

PageColorStyle.Light	Light color pattern
PageColorStyle.Dark	Dark color pattern

LanguageStyle

LanguageStyle Value	Description
LanguageStyle.AUTO	Auto
LanguageStyle.ENGLISH	English
LanguageStyle.SIMPLIFIED_CHINESE	Simplified Chinese
LanguageStyle.TRADITIONAL_CHINESE	Traditional Chinese
LanguageStyle.CUSTOMIZE_LANGUAGE	Custom language

AuthUiConfig

Parameter configuration of custom liveness detection and face comparison UI.

Type	Name	Description	Default Value
VideoSize	videoSize	Resolution during liveness detection and face comparison	480P
boolean	isShowCountdown	Whether to display the countdown control	true
boolean	isShowErrorDialog	Whether to display the error dialog	true
int	authLayoutResId	Custom layout <code>resID</code> exported from FaceID, which is -1 by default if not adjusted.	-1
int	feedBackErrorColor	Color of exception <code>tips</code> (format: 0xFFFFFFFF), which is -1 by default if not adjusted.	-1
int	feedBackTxtColor	Color of success <code>tips</code> (format: 0xFFFFFFFF), which is -1 by default if not adjusted.	-1
int	authCircleErrorColor	Color of background round frame for incorrect action (format: 0xFFFFFFFF), which is -1 by default if not adjusted.	-1
int	authCircleCorrectColor	Color of background round frame of correct action (format: 0xFFFFFFFF), which is -1 by default if not adjusted.	-1

int	authLayoutBgColor	Color of liveness detection and face comparison UI background (format: 0xFFFFFFFF), which is -1 by default if not adjusted.	-1
-----	-------------------	---	----

VideoSize

Enumerated values of resolutions supported by eKYC.

VideoSize Value	Description
VideoSize.SIZE_480P	480P
VideoSize.SIZE_720P	720P

HuiYanOsAuthResult

The result type corresponding to the callback for successful liveness detection and face comparison.

Type	Name	Description	Default Value
String	token	The token used in the current liveness detection and face comparison process.	Empty
String	bestImage	Base64-encoded data of the best frame image for liveness detection and face comparison.	Empty

You will need to use the token to call the [GetFacelResultIntl](#) API to pull the liveness detection and face comparison result.

HuiYanOsAuthCallBack

This is the callback API for the liveness detection and face comparison process.



```
/**
 * Result callback
 *
 * @author jerrydong
 * @since 2022/6/10
 */
public interface HuiYanOsAuthCallBack {

    /**
     * Successful liveness detection and face comparison
     *
     */
}
```



```
    * @param authResult Result
    */
    void onSuccess(HuiYanOsAuthResult authResult);

    /**
     * Failed liveness detection and face comparison
     *
     * @param errorCode Error code
     * @param errorMsg Error message
     * @param token Token used in this liveness detection and face comparison process
     */
    void onFail(int errorCode, String errorMsg, String token);
}
```

HuiYanAuthEventCallback

This callback is used to listen for key events during liveness detection and face comparison. You can use the UI with custom layout to [bind events \(see the document about custom capabilities\)](#).



```
/**
 * Callback for liveness detection and face comparison event in FaceID SDK
 */
public interface HuiYanAuthEventCallBack {

    /**
     * Callback for event notification when liveness detection and face comparison
     *
     * @param tipsEvent Key `tips` event
     */
    void onAuthTipsEvent(HuiYanAuthTipsEvent tipsEvent);
}
```

```
/**
 * Liveness detection and face comparison event
 *
 * @param authEvent authEvent
 */
void onAuthEvent(HuiYanAuthEvent authEvent);

/**
 * Callback for the creation of the authenticated main view
 *
 * @param authView
 */
void onMainViewCreate(View authView);

/**
 * Callback for UI repossession
 */
void onMainViewDestroy();

}
```

Error Codes

Below are the error codes in the eKYC SDK (global edition) in failure callbacks and their meaning:

Error Codes	Error Code	Error Description
HY_NETWORK_ERROR	210	Network request exception.
HY_LOCAL_REF_FAILED_ERROR	211	Check failed during local SDK initialization. The common exceptions are license file nonexistence or license expiration.
HY_USER_CANCEL_ERROR	212	The user actively cancels the liveness detection and face comparison process.
HY_INNER_ERROR_CODE	213	An internal exception of the SDK occurred, causing the liveness detection and face comparison process to be stopped.
HY_DO_NOT_CHANGE_ERROR	214	Applications are switched during liveness detection and face comparison, causing the process to be stopped.
HY_CAMERA_PERMISSION_ERROR	215	An exception occurred while getting the

		camera.
HY_INIT_SDK_ERROR	216	The liveness detection and face comparison method is directly called before the "init()" method is called.
HY_VERIFY_LOCAL_ERROR	217	Local face detection failed.
HY_PERMISSION_CHECK_ERROR	218	The permissions required by the local SDK are insufficient.
HY_APP_STOP_ERROR	219	If <code>reflectSequence</code> of <code>startAuthByLightData</code> is <code>null</code> , you stopped the liveness detection and face comparison process.
HY_CHECK_LIVE_DATA_ERROR	220	Failed to verify the light sequence parameter.
HY_INITIALIZATION_PARAMETER_EXCEPTION	221	An exception occurred while directly calling the method for setting light sequence parameters without getting the device configuration.
HY_VERIFY_LOCAL_TIME_OUT	222	Local motion detection timed out.
HY_PREPARE_TIME_OUT	223	Preparation timed out (between camera launch and the first detection of face).
HY_CHECK_PERMISSION_ERROR	224	Failed to apply for the camera permission inside the SDK.

iOS API Description

Last updated : 2023-06-13 11:25:50

API description

Integration with the eKYC SDK mainly involves the following classes: `HuiYanOSKit` (API class), `HuiYanOsConfig` (configuration parameter class), and `HuiYanOKitSuccCallback` and `HuiYanOKitFailCallback` (result and callback classes).

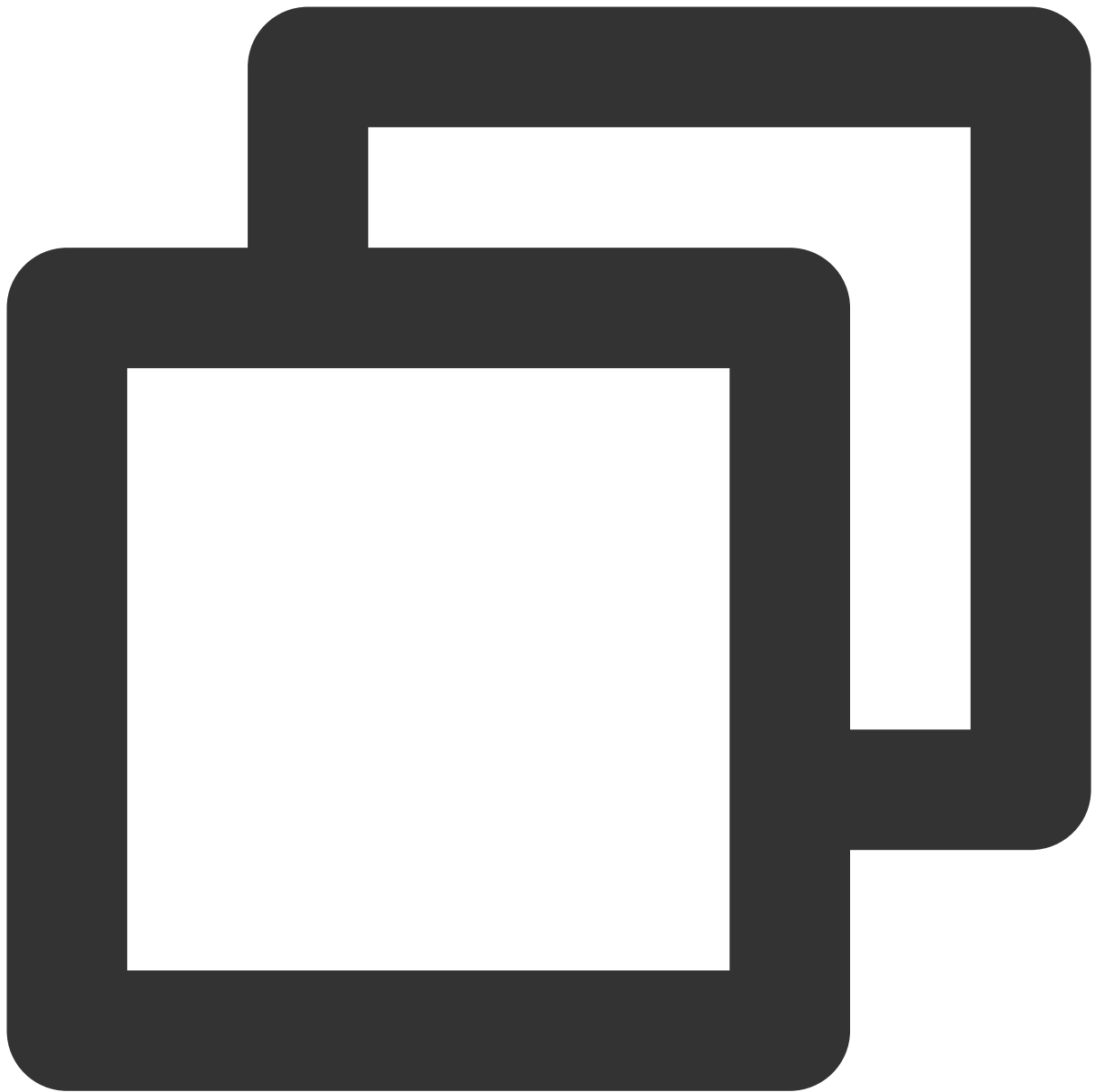
HuiYanOSKit

API	Feature Description
<code>startHuiYaneKYC()</code>	Starts the liveness detection and face comparison process in the FaceID SDK.
<code>release()</code>	Releases FaceID SDK resources.

`release()`

Feature description:

This API is used to release eKYC SDK resources.

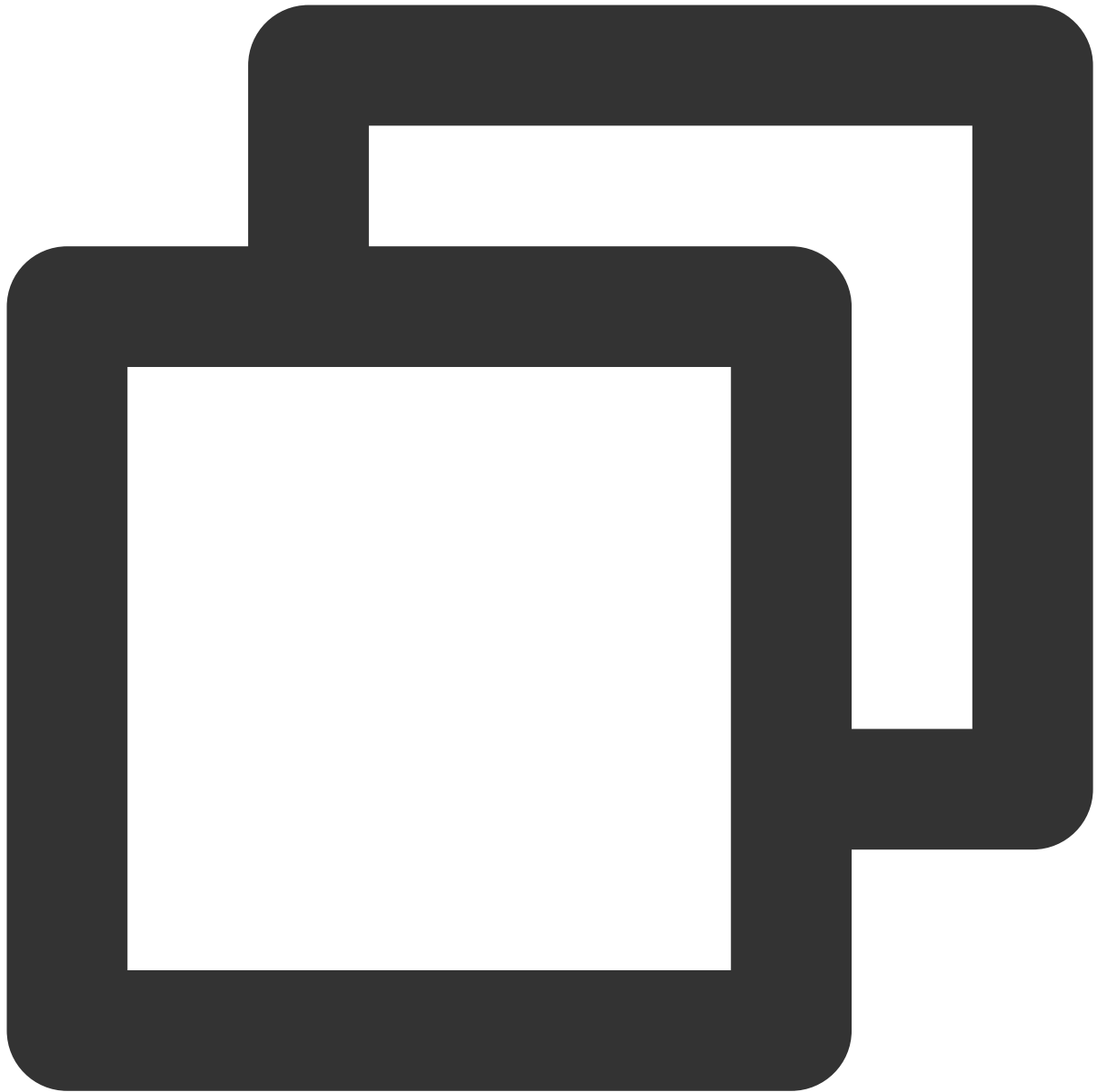


```
+ (void)release;
```

startHuiYanKYC

Feature description:

This API is used to start and configure the liveness detection and face comparison SDK and call back the result or error.



```
/// Start the liveness detection and face comparison process
/// @param faceToken token
/// @param kitConfig SDK configuration
/// @param succCallback Callback for successful liveness detection and face comparison
/// @param failCallback Callback for failed liveness detection and face comparison
- (void)startHuiYanKYC:(NSString *)faceToken withConfig:(HuiYanOsConfig *)kitConfig
    witSuccCallback:(HuiYanOKitSuccCallback) succCallback
    withFailCallback:(HuiYanOKitFailCallback) failCallback;
```

Input parameters:

--	--	--

Type	Parameter	Description
NSString	faceToken	Liveness detection and face comparison process token
HuiYanOsConfig	kitConfig	SDK configuration class
HuiYanOKitSuccCallback	succCallback	Callback for successful liveness detection and face comparison
HuiYanOKitFailCallback	failCallback	Callback for failed liveness detection and face comparison

HuiYanOsAuthResult

Type	Parameter	Description
NSString	faceToken	Liveness detection and face comparison process token
NSString	bestImage	Base64-encoded best frame image returned upon success if best frame return is enabled

You will need to use the token to call the **GetFaceIdResultIntl** API to pull the liveness detection and face comparison result.

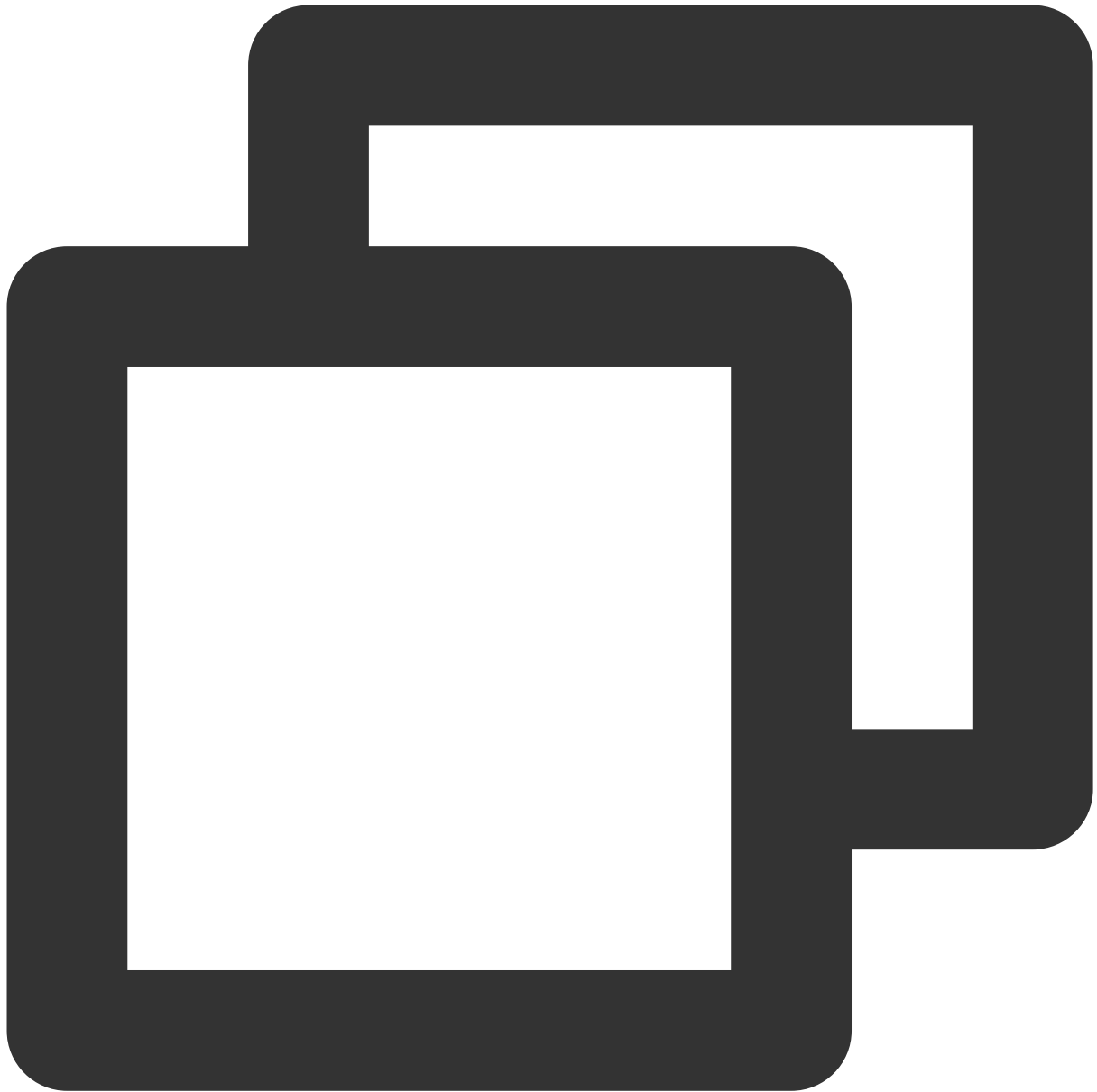
HuiYanOsConfig

`HuiYanOsConfig` is the configuration entity class during eKYC SDK startup, which mainly contains the following attributes:

Type	Name	Description	Default Value
NSString	authLicense	Name of the license file applied for by client for liveness detection and face comparison authorization	Empty
long	authTimeOutMs	Liveness detection and face comparison timeout period	10000 ms (10s)
long	prepareTimeoutMs	Detection timeout period in the preparation stage	0
HYShowTimeOutMode	showTimeOutMode	Stage in which the countdown is displayed	HYShowTimeOutMod
BOOL	isDeleteVideoCache	Whether to delete the local cache of the liveness detection and face comparison video	YES

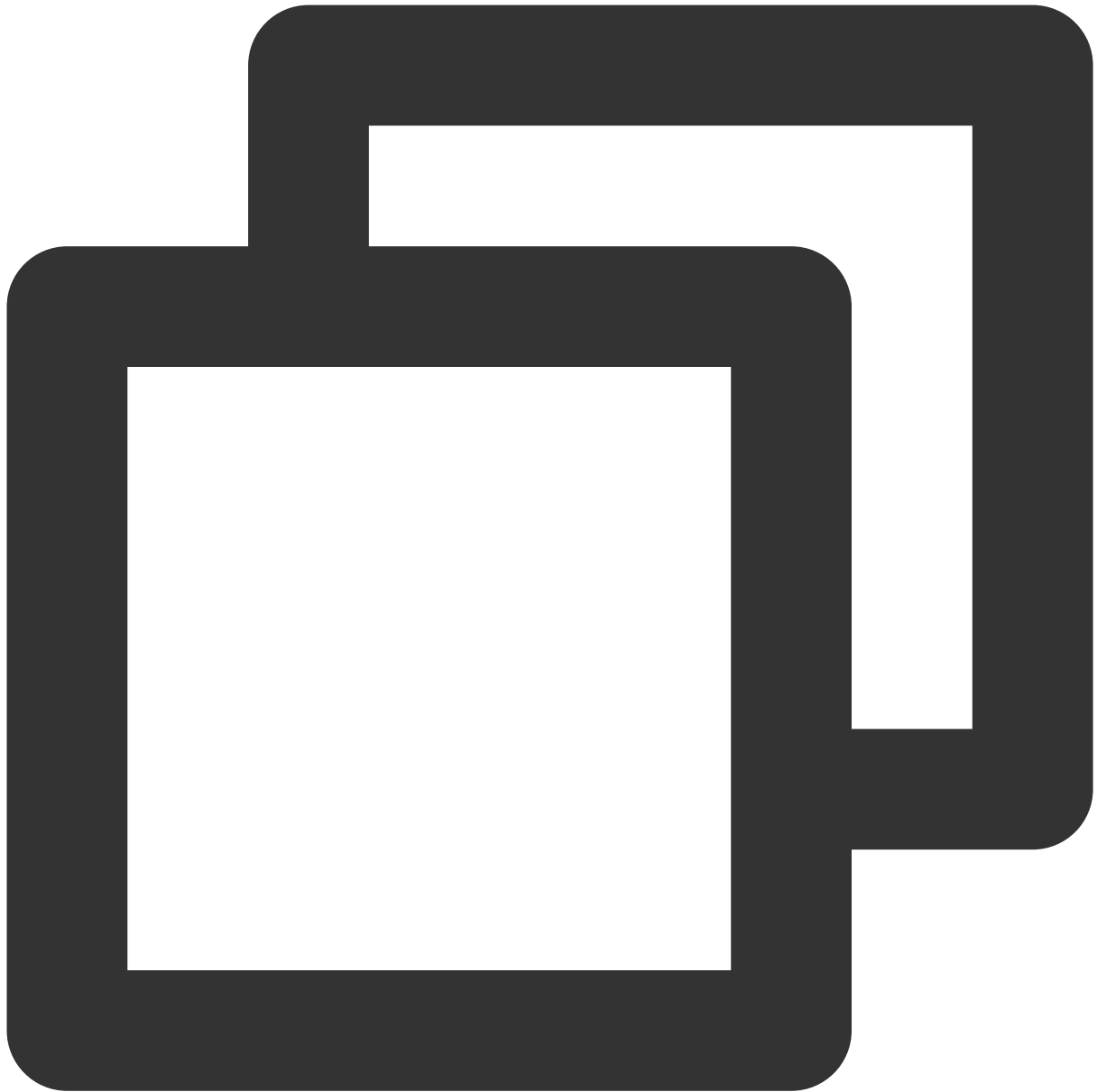
BOOL	iShowTipsPage	Whether to display the guide page	No
NSString	userUIBundleName	Custom UI bundle filename; for example, set <code>UserUIBundle</code> for <code>UserUIBundle.bundle</code>	nil
NSString	userLanguageFileName	Custom languageBundle name; for example, set <code>UseLanguage</code> for <code>UseLanguage.bundle`</code> .	nil
NSString	userLanguageBundleName	Custom local file name for internationalization; for example, set <code>en</code> for <code>en.lproj</code> .	nil
LanguageType	languageType	Text language settings inside the SDK	DEFAULT
BOOL	isGetBestImg	Whether to get the best frame image	No
NSString	setLanguageFileName	Language file directory name added in <code>HuiYanSDKUI.bundle</code> has the highest priority by default	nil

HuiYanOKitSuccCallback



```
/**
 * Callback for successful liveness detection and face comparison
 *
 * @param authResult Liveness detection and face comparison result
 * @param reserved Reserved
 */
typedef void (^HuiYanOKitSuccCallback)(HuiYanOsAuthResult * _Nonnull authResult, id
```

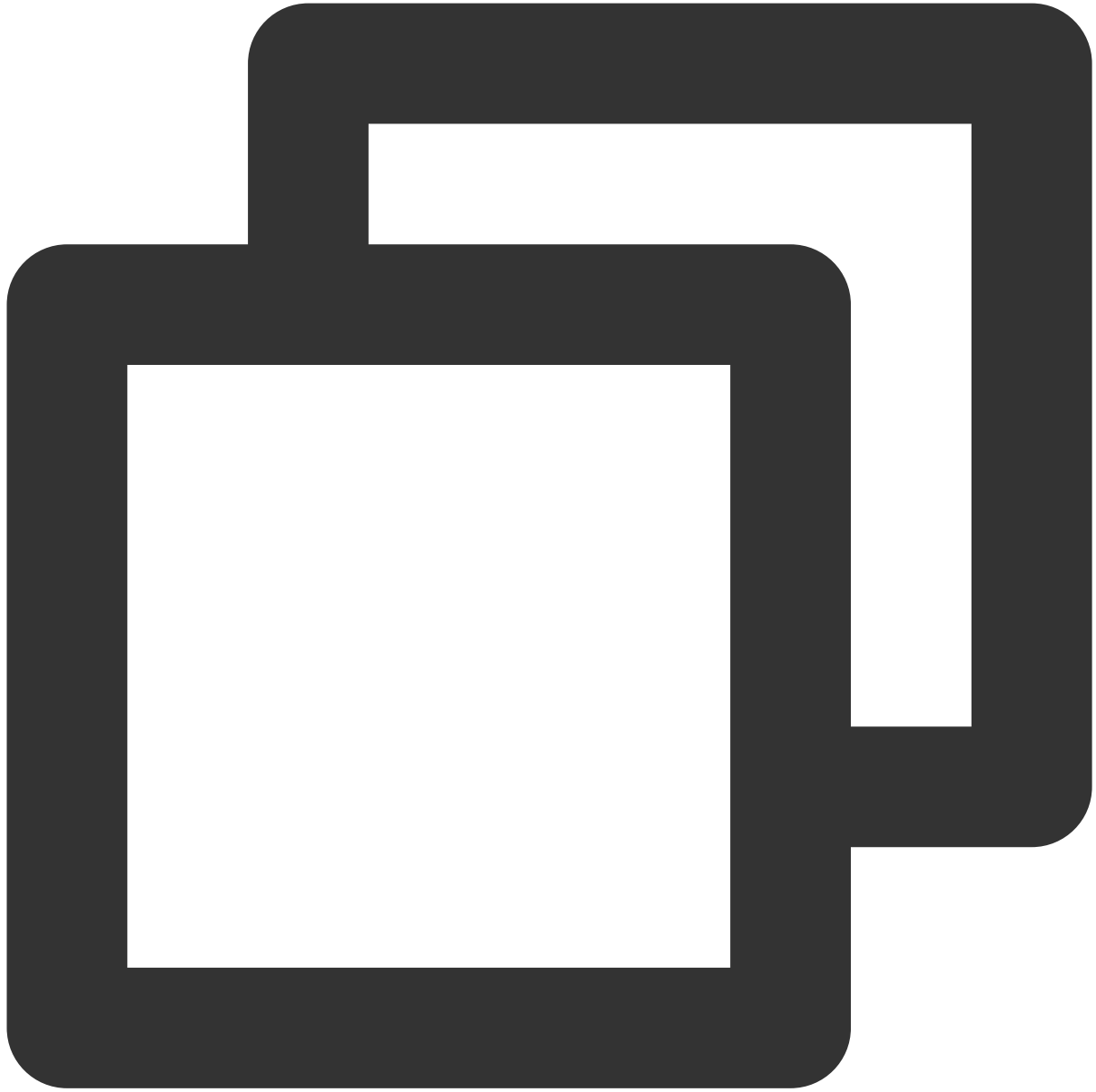
HuiYanOKitFailCallback



```
/**
 * Callback for failed liveness detection and face comparison
 *
 * @param errCode Error code
 * @param errMsg Error message
 * @param reserved Reserved
 */
typedef void (^HuiYanOKitFailCallback)(int errCode, NSString * _Nonnull errMsg ,id
```

LanguageType

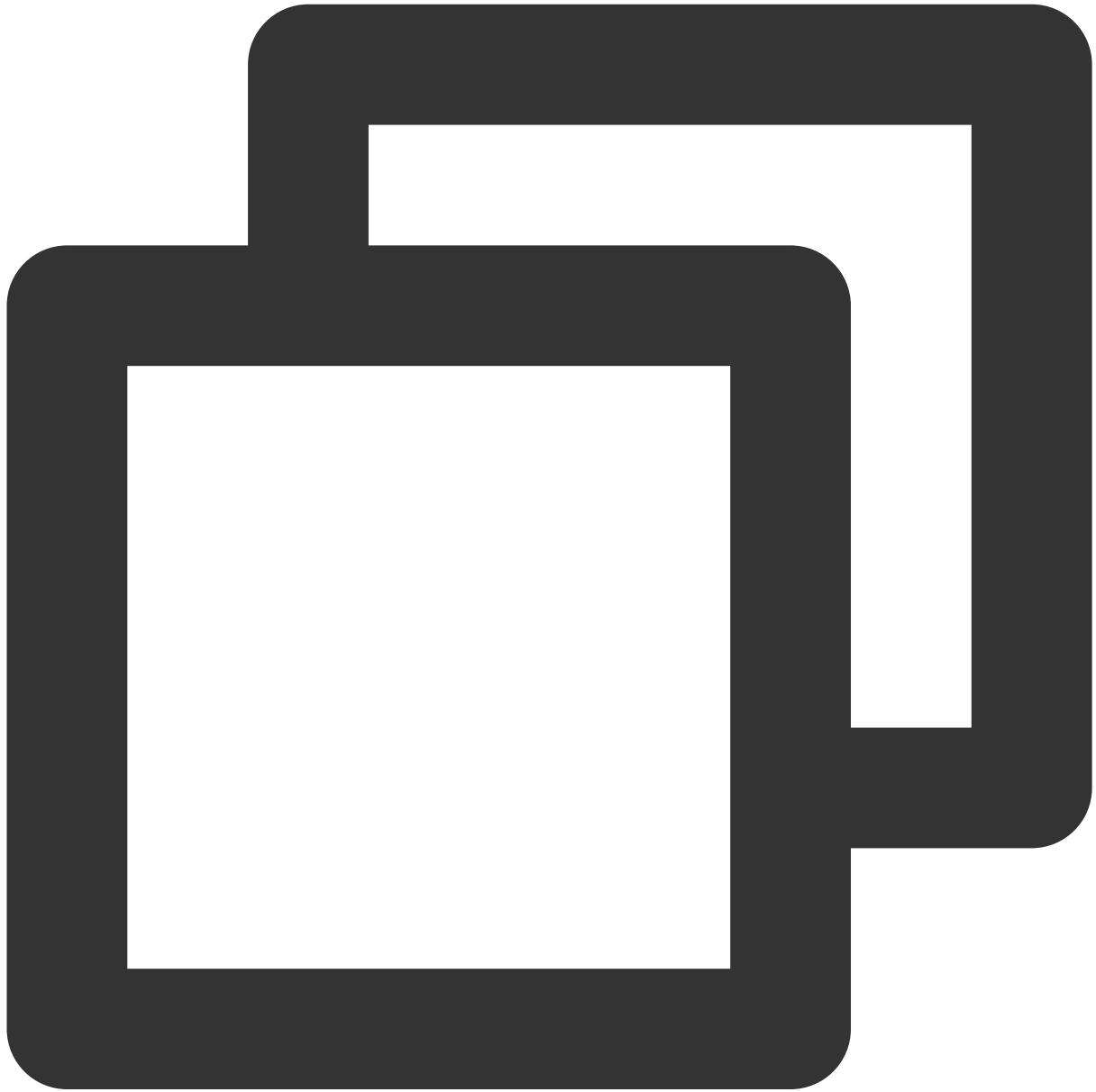
Internationalization strings included in the SDK



```
typedef enum : NSUInteger {  
    DEFAULT = 0, //Auto  
    ZH_HANS, //Simplified Chinese  
    ZH_HANT, //Traditional Chinese  
    ZH_HK, //Traditional Chinese (Hong Kong)  
    ZH_TW, //Traditional Chinese (Taiwan)  
    EN, //English  
    MS, //Malaysian  
    RU, //Russian
```

```
JA, //Japanese
CUSTOMIZE_LANGUAGE, //Custom language
} LanguageType;
```

HYShowTimeOutMode



```
typedef NS_OPTIONS(int, HYShowTimeOutMode) {
    HYShowTimeOutMode_TIMEOUT_HIDDEN = 1 << 0, // Hide the countdown in all stages
    HYShowTimeOutMode_PREPARE = 1 << 1, // Preparation stage countdown
    HYShowTimeOutMode_ACTION = 1 << 3, // Action stage countdown
};
```

SDK Error Codes (iOS)

Error Codes	Error Code	Error Description
HY_SUCCESS	0	Successful
HY_INITIALIZATION_PARAMETER_EXCEPTION	210	Parameter initialization exception
HY_BUNDLE_CONFIGURATION_EXCEPTION	211	Bundle configuration exception
HY_YTSDK_CONFIGURATION_EXCEPTION	212	YouTu configuration exception
HY_PLEASE_CALL_FIRST_INIT_API	213	Call the API for initialization first
HY_SDK_AUTH_FAILED	214	SDK authorization failure
HY_USER_VOLUNTARILY_CANCELED	215	Manually canceled by user
HY_YTSDK_LOCAL_AUTH_FAILED	216	Local face detection failure of SDK
HY_CAMERA_OPEN_FAIL	217	Failed to enable the camera
HY_DONOT_SWITCH_APPS	218	Do not switch the application during liveness detection and face comparison
HY_CAMEREA_PERMISSION_EXCEPTION	219	Camera permission exception
HY_SDK_VEDIO_CUT_EXCEPTION	220	Failed to clip the video
HY_LIGHT_DATA_FORMAT_EXCEPTION	221	Incorrect light data format

SDK Custom Capabilities

Android Custom Capabilities

Last updated : 2023-06-13 11:27:15

This document introduces the custom capabilities of the eKYC SDK (global edition).

I. Custom UI

Customizing layout

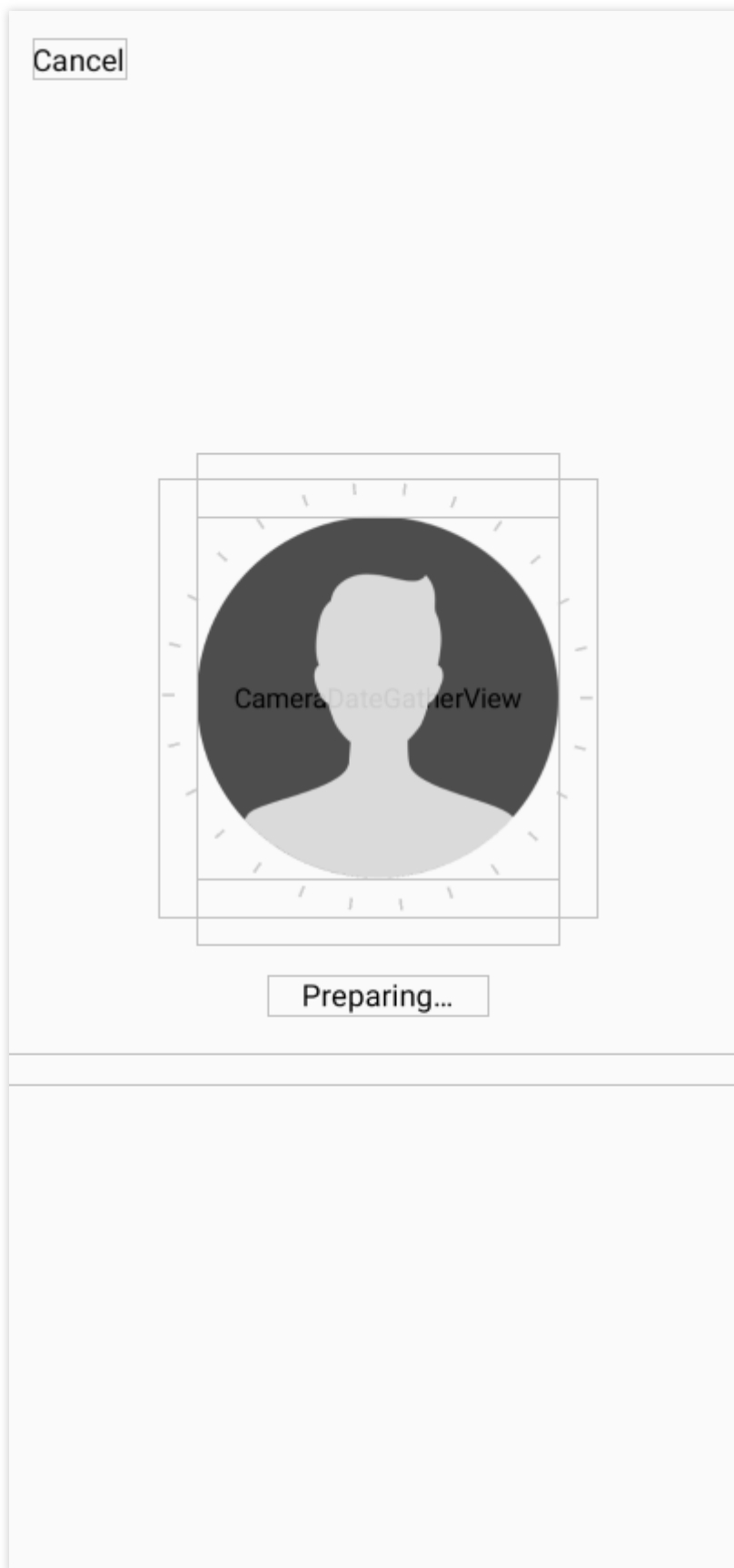
The eKYC SDK supports custom UI by using `AuthUiConfig` (see the API description document) or passing in `Layout resId`.

The usage is as follows:



```
AuthUiConfig authConfig = new AuthUiConfig();  
authUiConfig.setAuthLayoutResId(R.layout.demo_huiyan_fragment_authing);  
authUiConfig.setAuthCircleCorrectColor(resources.getColor(R.color.demo_blue));  
huiYanOsConfig.setAuthUiConfig(authUiConfig);
```

The following figure shows the default layout:

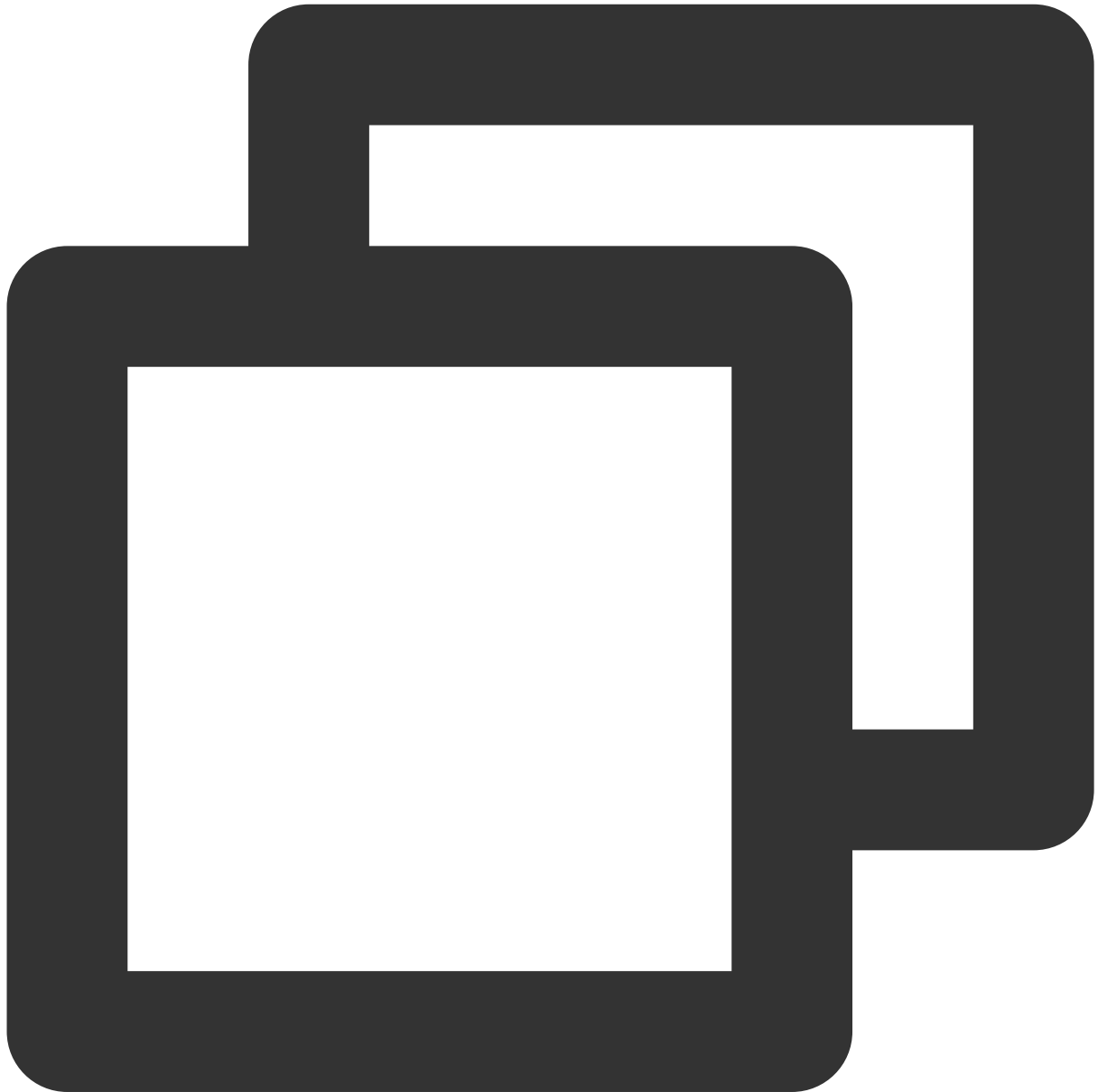


The default layout is as shown above, where the positions of all controls can be adjusted by modifying the `Layout.xml` file. For more information, see the demo's `demo_huiyan_fragment_authing.xml` file.

Note:

The `View type` and the corresponding `android:id` in `demo_huiyan_fragment_authing.xml` are involved in UI event binding, so **do not modify them**.

The default layout provided here is as follows:



```
<?xml version="1.0" encoding="utf-8"?>
<com.tencent.could.huiyansdk.view.HuiYanReflectLayout xmlns:android="http://schemas
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:id="@+id/txy_auth_layout_bg"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
```

```
<!-- Cancel button -->
<TextView
    android:id="@+id/txy_cancel_txt_btn"
    android:text="@string/txy_cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:textColor="@color/txy_black"
    android:textSize="16sp"
    android:layout_marginTop="@dimen/txy_title_margin_top"
    android:layout_marginStart="@dimen/txy_protocol_margin_size"
/>

<!-- Countdown display control -->
<TextView
    android:id="@+id/txy_count_down_txt_view"
    android:text="@string/txy_count_down_txt"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:textSize="16sp"
    android:textColor="@color/txy_black"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/txy_title_margin_top"
    android:layout_marginRight="@dimen/txy_protocol_margin_size"
    android:visibility="gone"
/>

<!-- Camera preview frame (if it is 720p, the height in the code will be automa
<com.tencent.could.huiyansdk.view.CameraDateGatherView
    android:id="@+id/txy_camera_gather_view"
    android:layout_width="@dimen/txy_auth_head_size"
    android:layout_height="258dp"
    android:background="@android:color/transparent"
    android:layout_marginBottom="@dimen/txy_auth_view_move"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<!-- General background, which is a ring with vertical lines -->
<com.tencent.could.huiyansdk.view.CommonAuthBackView
    android:id="@+id/txy_auth_common_background_views"
    android:layout_width="230dp"
    android:layout_height="230dp"
```

```
        android:layout_marginBottom="@dimen/txy_auth_view_move"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
    />

<!-- Profile photo to display -->
<ImageView
    android:id="@+id/txy_camera_prepare_img"
    app:srcCompat="@drawable/txy_prepare_face_head_white"
    android:layout_width="@dimen/txy_auth_head_size"
    android:layout_height="@dimen/txy_auth_head_size"
    android:layout_marginBottom="@dimen/txy_auth_view_move"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />

<!-- Frontend animation view -->
<com.tencent.could.huiyansdk.view.LoadingFrontAnimatorView
    android:id="@+id/txy_auth_loading_front_animator_view"
    android:visibility="gone"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginBottom="@dimen/txy_auth_view_move"
    android:layout_width="@dimen/txy_auth_head_size"
    android:layout_height="@dimen/txy_auth_head_size"
/>

<!-- Prompt display page -->
<TextView
    android:id="@+id/txy_auth_feed_back_txt"
    app:layout_constraintTop_toBottomOf="@id/txy_auth_common_background_views"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:textColor="@color/txy_black"
    android:layout_marginTop="@dimen/txy_protocol_line_space"
    android:text="@string/txy_face_preparing3"
    android:textSize="18sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

<!-- Prompt control for additional warning message -->
<TextView
```

```
android:id="@+id/txy_auth_feed_back_extra_tip_txt"
android:textSize="14sp"
android:textColor="@color/txy_black"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center_horizontal"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"
android:layout_marginTop="20dp"
app:layout_constraintTop_toBottomOf="@id/txy_auth_feed_back_txt"
/>
```

```
<!-- Content prompt for liveness detection and face comparison -->
```

```
<TextView
    android:id="@+id/txy_auth_tips_txt"
    android:textSize="14sp"
    android:textColor="@color/txy_black"
    android:paddingHorizontal="25dp"
    android:layout_marginHorizontal="25dp"
    android:layout_marginBottom="35dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

```
</com.tencent.cloud.huiliansdk.view.HuiYanReflectLayout>
```

Binding events for custom layout

The previous section describes how to customize the UI layout. This section describes how to bind events to the controls added to the custom UI layout to meet your specific needs.

When the liveness detection and face comparison UI is created, `onMainViewCreate(View authView)` of `HuiYanAuthEventCallBack` will be called. When the liveness detection and face comparison UI is to be terminated, the `onMainViewDestroy()` method will be called back, and you can customize the processing logic in the corresponding lifecycle.



```
// Set a callback listener for key events in FaceID
HuiYanOsApi.setAuthEventCallBack(new HuiYanAuthEventCallBack() {
    @Override
    public void onAuthTipsEvent(HuiYanAuthTipsEvent tipsEvent) {
        Log.e(TAG, "current is : " + tipsEvent);
    }

    @Override
    public void onMainViewCreate(View authView) {
        if (authView == null) {
            return;
        }
    }
});
```

```
    }  
    // Get the added controls and register custom events  
    Button findBtn = authView.findViewById(R.id.add_view_offer_button);  
    if (findBtn != null) {  
        findBtn.setOnClickListener(view -> {  
            Log.e(TAG, "click test button!");  
        });  
    }  
}  
  
@Override  
public void onMainViewDestroy() {  
    Log.e(TAG, "onMainViewDestroy");  
}  
});
```

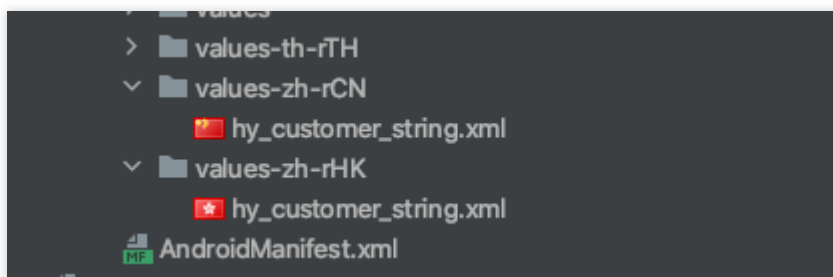
Based on the callback of `onMainViewCreate` and the callback method of `onMainViewDestroy`, you can bind events to the added UI controls to achieve the desired effect.

II. Custom Prompt and Language

Customizing prompt

If you want to modify a prompt or add a language file, follow the instructions below. The eKYC SDK provides a translation file `hy_customer_string.xml`, which contains all the configuration text that can be modified.

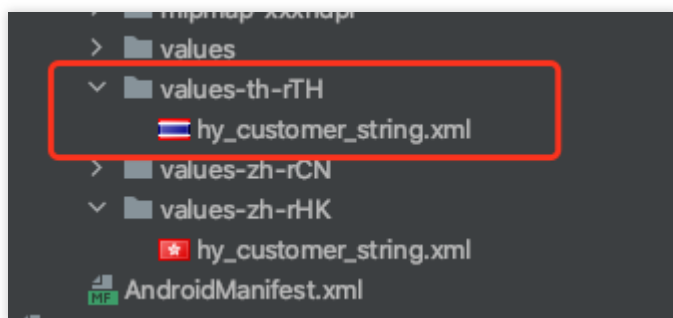
1. Open the project in the main module (which integrates the eKYC SDK).
2. Add the `hy_customer_string.xml` file to the corresponding language folder.
3. Modify the text content that needs to be customized.
4. After being packaged, the modified content will automatically overwrite the original content.



Adding a language

To add a language, perform the following steps:

1. Add the corresponding language folder to the project in the main module (which integrates the eKYC SDK).
2. Copy the `hy_customer_string.xml` file to the language folder and modify the value content.



3. Specify the target language code in the code (with Thai as an example).



```
huiYanOsConfig.setLanguageStyle(LanguageStyle.CUSTOMIZE_LANGUAGE);  
huiYanOsConfig.setLanguageCode("th-TH");
```

Language codes for Android

Some language codes for Android are provided for reference.

Language Code	Language - Country/Region
af-ZA	Afrikaans - South Africa

sq-AL	Albanian - Albania
ar-DZ	Arabic - Algeria
ar-BH	Arabic - Bahrain
ar-EG	Arabic - Egypt
ar-IQ	Arabic - Iraq
ar-JO	Arabic - Jordan
ar-KW	Arabic - Kuwait
ar-LB	Arabic - Lebanon
ar-LY	Arabic - Libya
ar-MA	Arabic - Morocco
ar-OM	Arabic - Oman
ar-QA	Arabic - Qatar
eu-ES	Basque - Basque
be-BY	Belarusian - Belarus
bg-BG	Bulgarian - Bulgaria
ca-ES	Catalan - Catalonia
zh-HK	Chinese - Hong Kong (China)
zh-MO	Chinese - Macao (China)
zh-CN	Chinese - China
zh-SG	Chinese - Singapore
zh-TW	Chinese - Taiwan (China)
zh-CHS	Simplified Chinese
zh-CHT	Traditional Chinese
hr-HR	Croatian - Croatia
cs-CZ	Czech - Czech Republic

da-DK	Danish - Denmark
div-MV	Dhivehi - Maldives
nl-BE	Dutch - Belgium
nl-NL	Dutch - Netherlands
en-AU	English - Australia
en-CA	English - Canada
en-ZA	English - South Africa
en-PH	English - Philippines
en-NZ	English - New Zealand
en-GB	English - UK
en-US	English - US
fa-IR	Persian - Iran
fi-FI	Finnish - Finland
fr-FR	French - France
fr-BE	French - Belgium
fr-MC	French - Monaco
fr-CH	French - Switzerland
gl-ES	Galician - Galicia
ka-GE	Georgian - Georgia
de-DE	German - Germany
de-LU	German - Luxembourg
de-CH	German - Switzerland
el-GR	Greek - Greece
gu-IN	Gujarati - India
he-IL	Hebrew - Israel

hi-IN	Hindi - India
hu-HU	Hungarian - Hungary
is-IS	Icelandic - Iceland
it-IT	Italian - Italy
ja-JP	Japanese - Japan
kk-KZ	Kazakh - Kazakhstan
kn-IN	Kannada - India
ko-KR	Korean - South Korea
lv-LV	Latvian - Latvia
lt-LT	Lithuanian - Lithuania
ms-BN	Malay - Brunei
ms-MY	Malay - Malaysia
mr-IN	Marathi - India
mn-MN	Mongolian - Mongolia
nn-NO	Nynorsk - Norway
pl-PL	Polish - Poland
pt-BR	Portuguese - Brazil
pt-PT	Portuguese - Portugal
ro-RO	Romanian - Romania
sa-IN	Sanskrit - India
ru-RU	Russian - Russia
sk-SK	Slovak - Slovakia
es-AR	Spanish - Argentina
es-ES	Spanish - Spain
sv-SE	Swedish - Sweden

th-TH	Thai - Thailand
tr-TR	Turkish - Türkiye
uk-UA	Ukrainian - Ukraine
ur-PK	Urdu - Pakistan
vi-VN	Vietnamese - Vietnam

iOS Custom Capabilities

Last updated : 2023-06-13 11:27:57

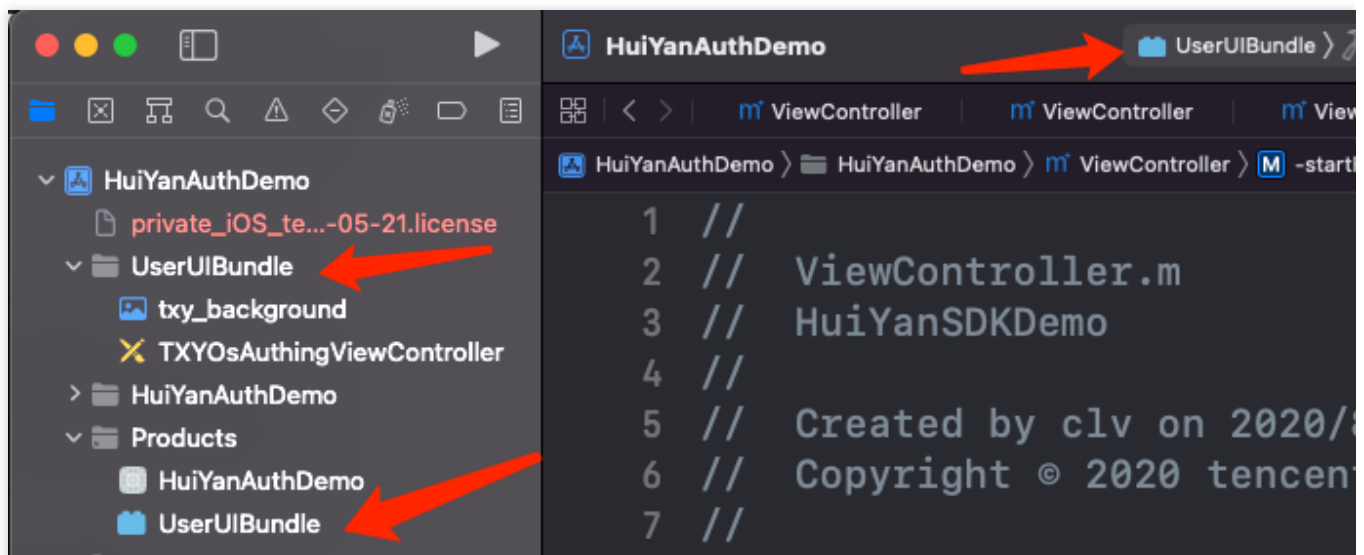
This document introduces the custom capabilities of the eKYC SDK (global edition).

I. Custom UI

This section describes how to use the custom UI to customize the recognition page and icons.

Building bundle

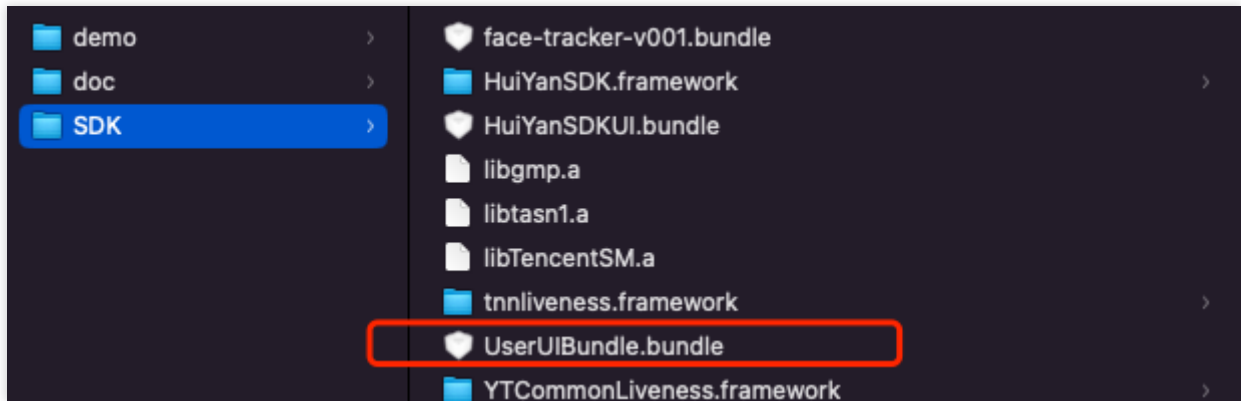
1. Decompress the SDK package, go to the demo directory, and open the HuiYanODemo project.
2. Switch the building scheme to `UserUIBundle` , and run `command+B` to build artifacts.



3. After `bundle` is built, right-click it to view the package content. Open `info.plist` , and delete the `Executable file` field. If this field is not deleted, artifacts cannot be published at AppStore, which causes an IPA upload error. You can also directly delete the `info.plist` file.

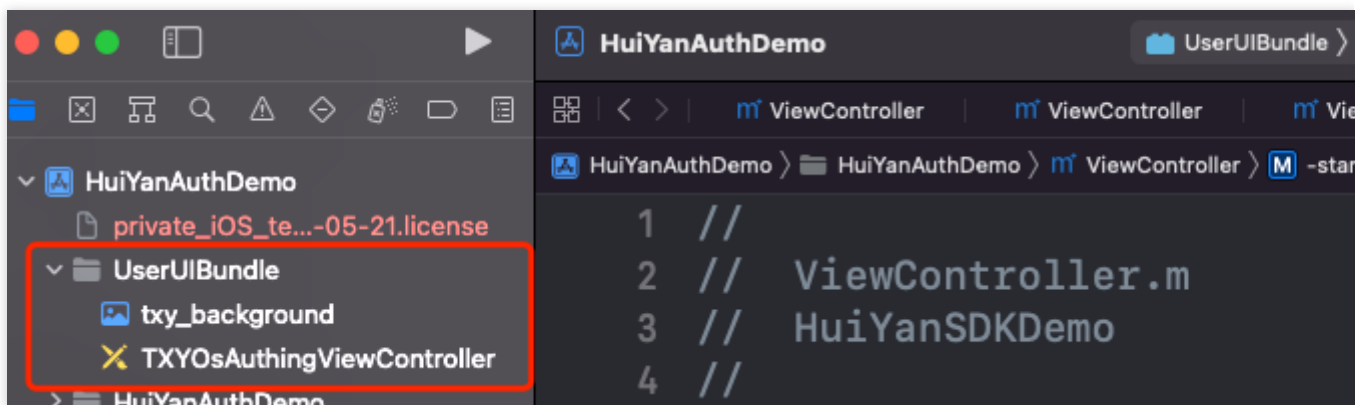
InfoDictionary version	String	6.0
Executable file	String	OcrSDK
DTCompiler	String	com.apple.compilers.llvm.clang.1_0

4. Directly replace `UserUIBundle.bundle` in the root directory (that is, SDK directory) with artifacts, and add all `lib` and `bundle` in this directory to your project. (Note: If the `bundle` content is not modified, you can directly use `bundle` in this directory without replacement.)

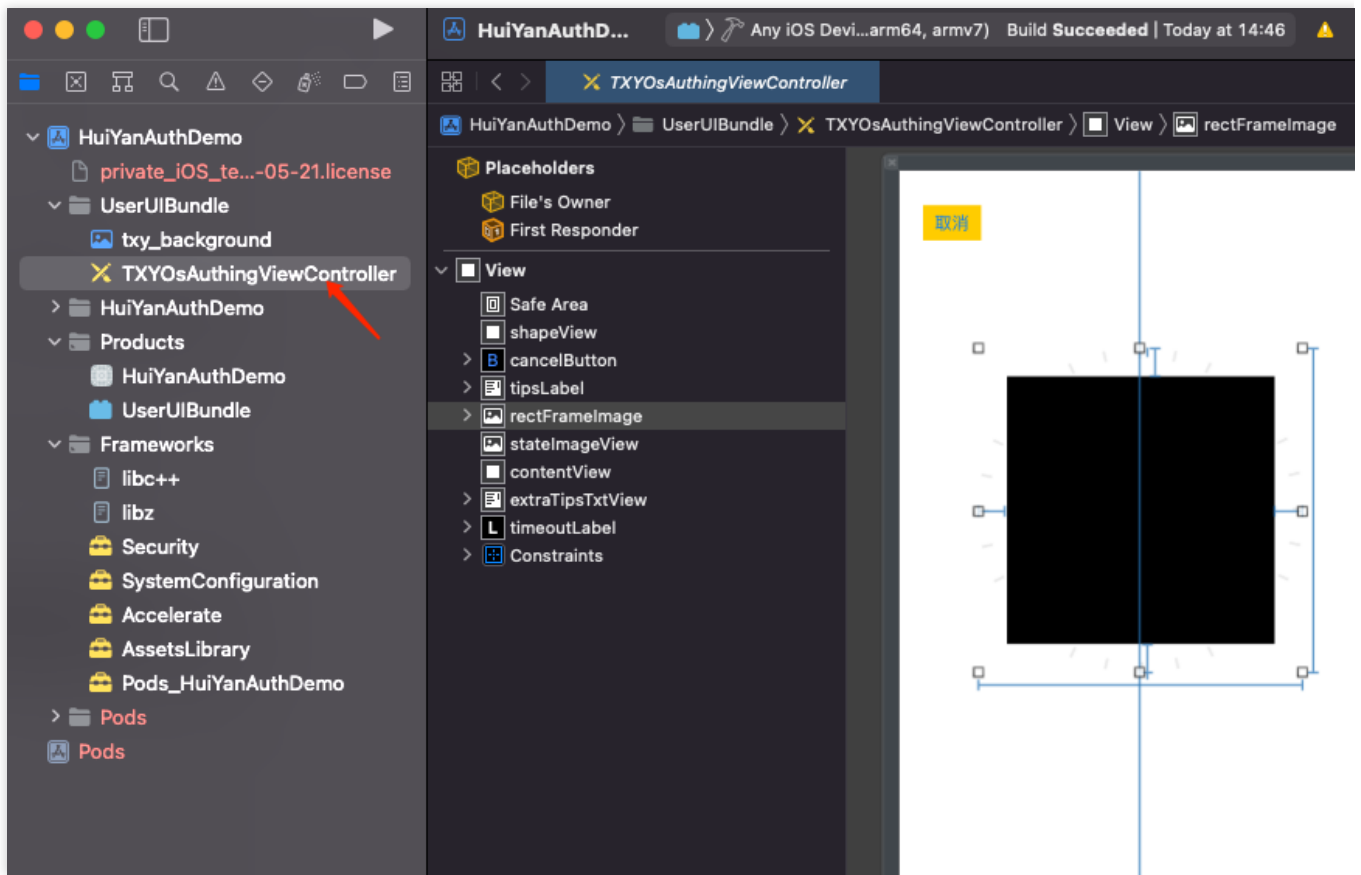


Customizing content

1. Icon: You can replace Icon to the following list, with the name unchanged.



2. xib layout adjustment: For the TXYOAuthingViewController recognition page, you can modify control layout in xib by adding static components, but not deleting components. The logic of this page is implemented by the internal .m file of SDK. You are allowed to modify the layout and add non-logical components only. (For example, add a background image.)



3. You can add settings to SDK by setting the `userUIBundleName` field.

After this setting, the UI bundle file with custom packaging will be used, and the

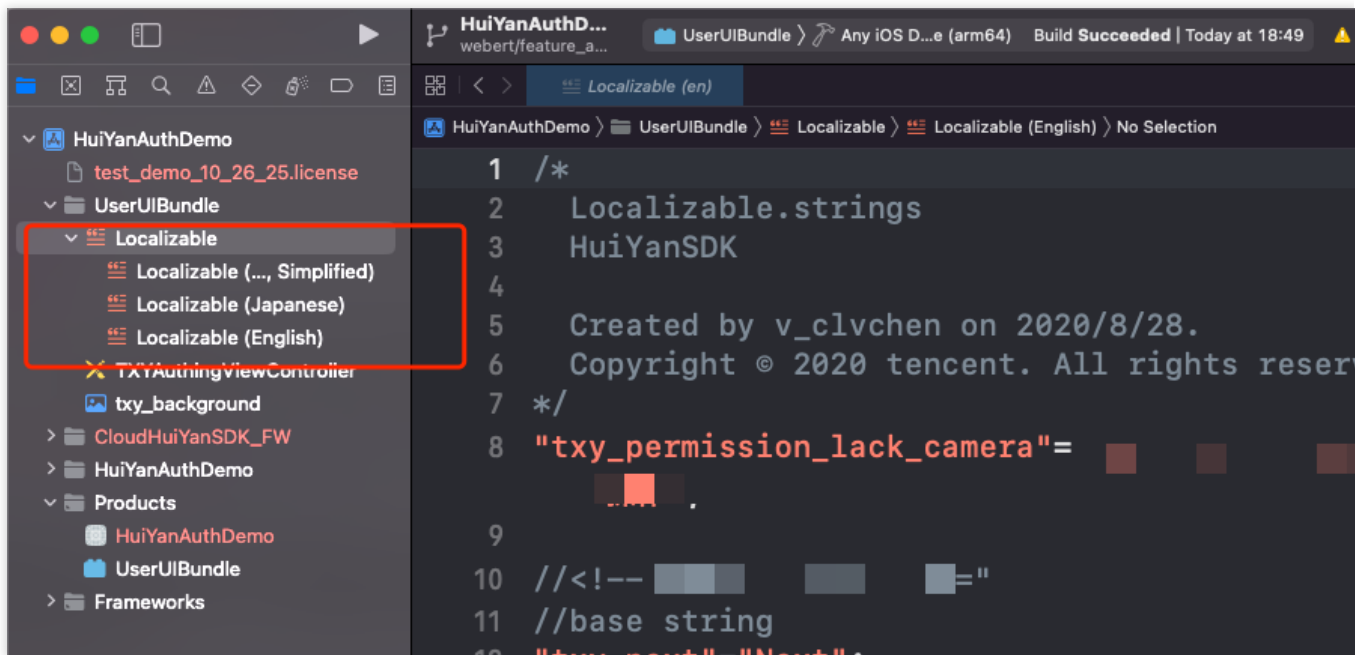
`TXYOAuthingViewController` layout file in it will be loaded.

If this layout file is not found, the default layout will be loaded.

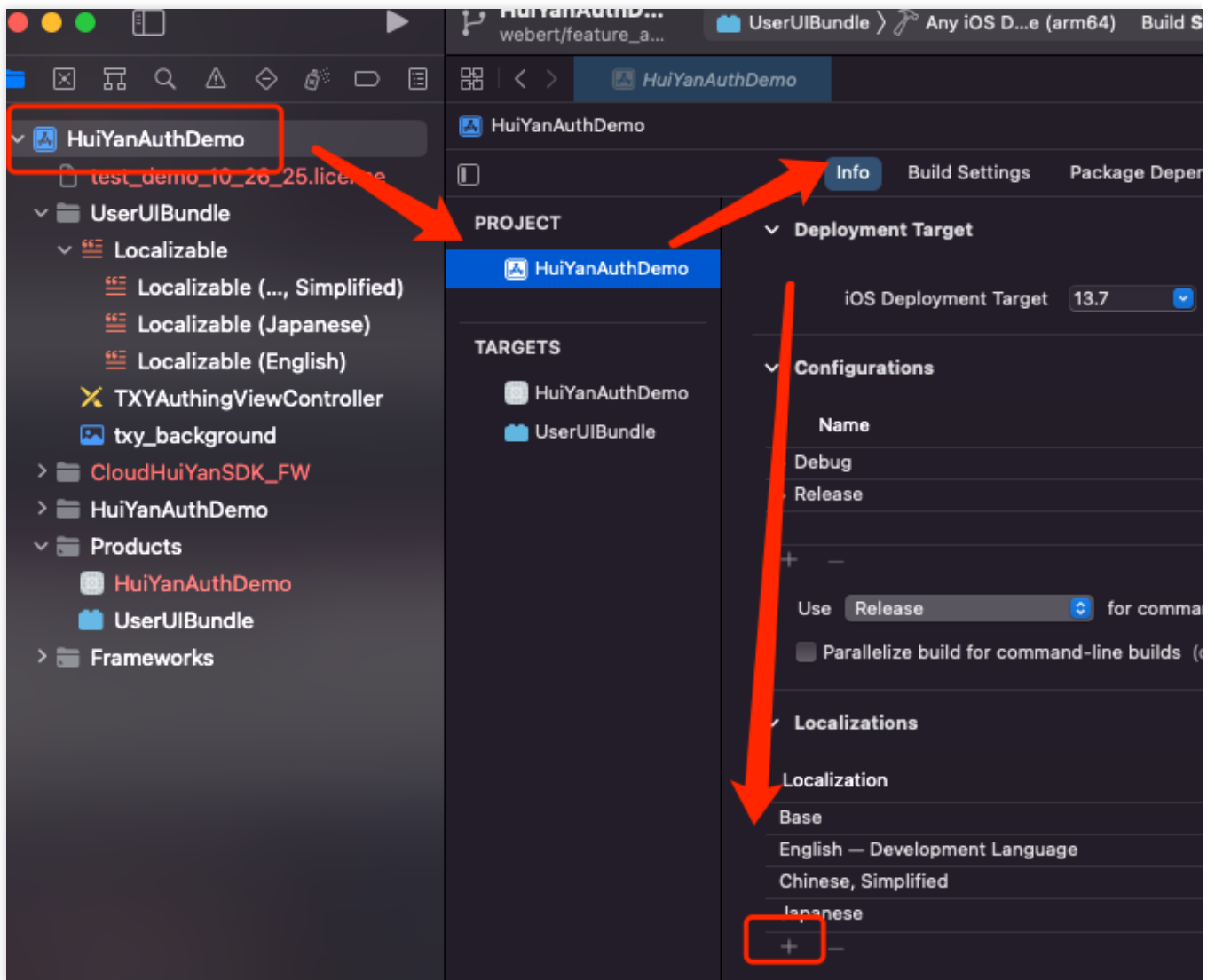
II. Custom language

Adding a custom language

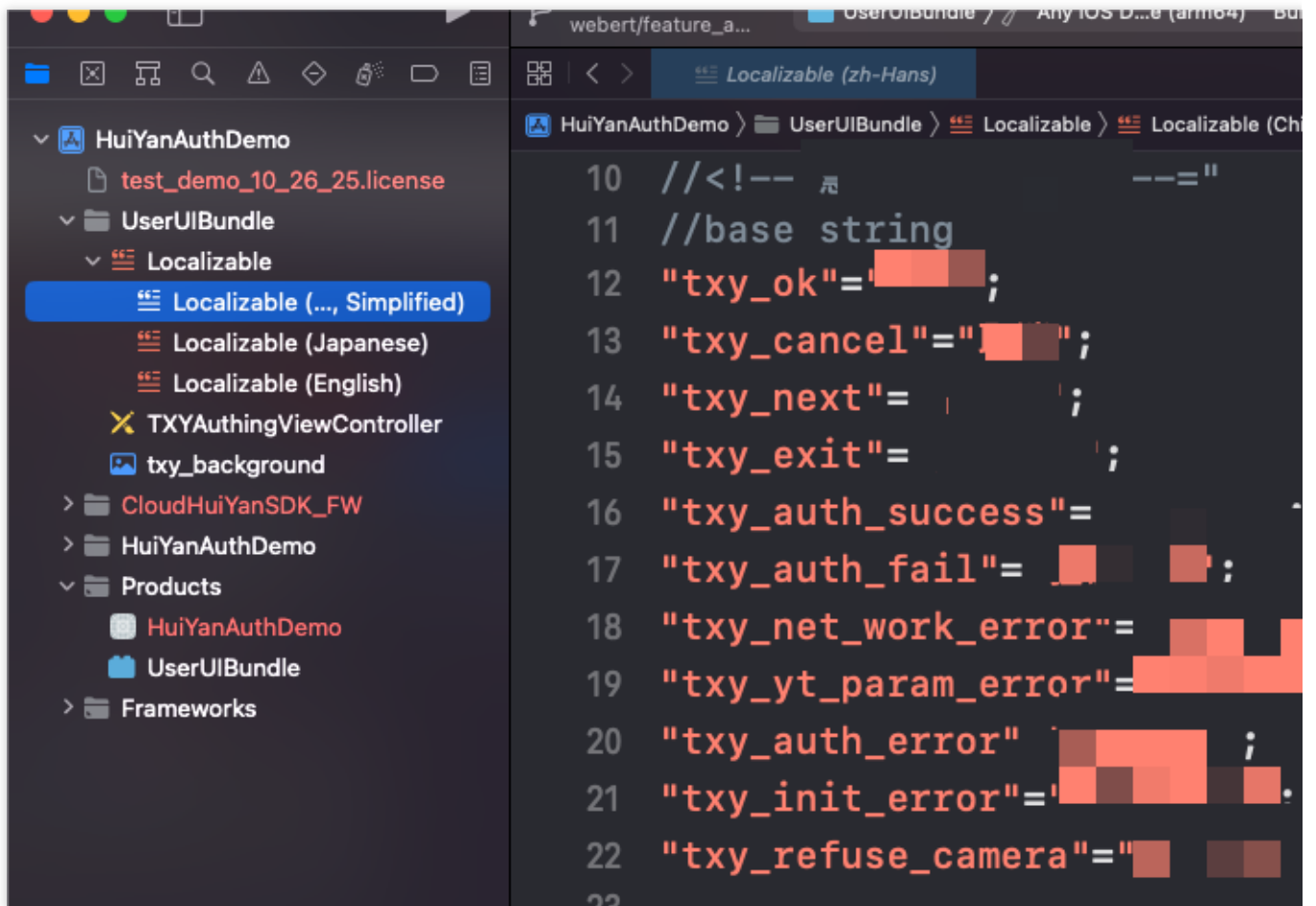
1. In the Demo, the `UserUIBundle` folder contains `Localizable`. As shown in the following figure, you can set supported languages on the right, and corresponding subfiles are displayed on the left. In the subfiles, multi-language mapping is done for the existing key character strings.



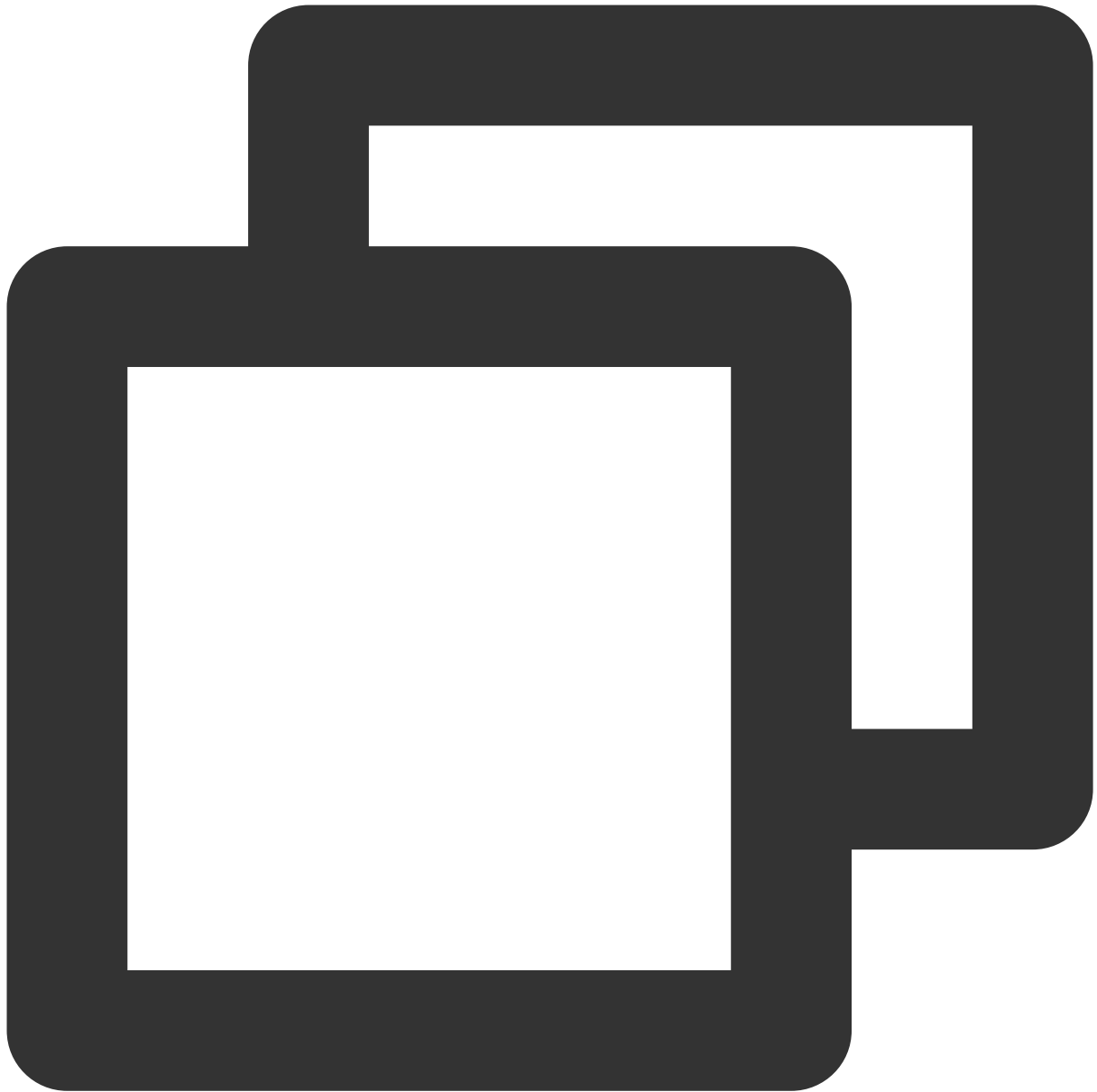
2. If there is no target language on the right, you can first add the corresponding language to the project settings, and then repeat step 1.



3. Perform translation mapping on the target file. The following example shows the mapping of simplified Chinese. To add mapping for other languages, keep `key` on the left unchanged and add translation on the right.

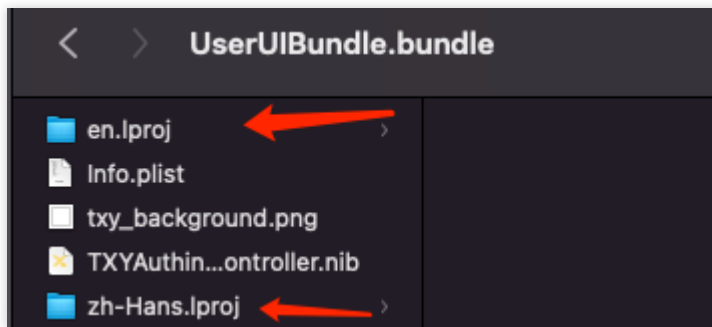


4. Settings in the SDK are as follows:



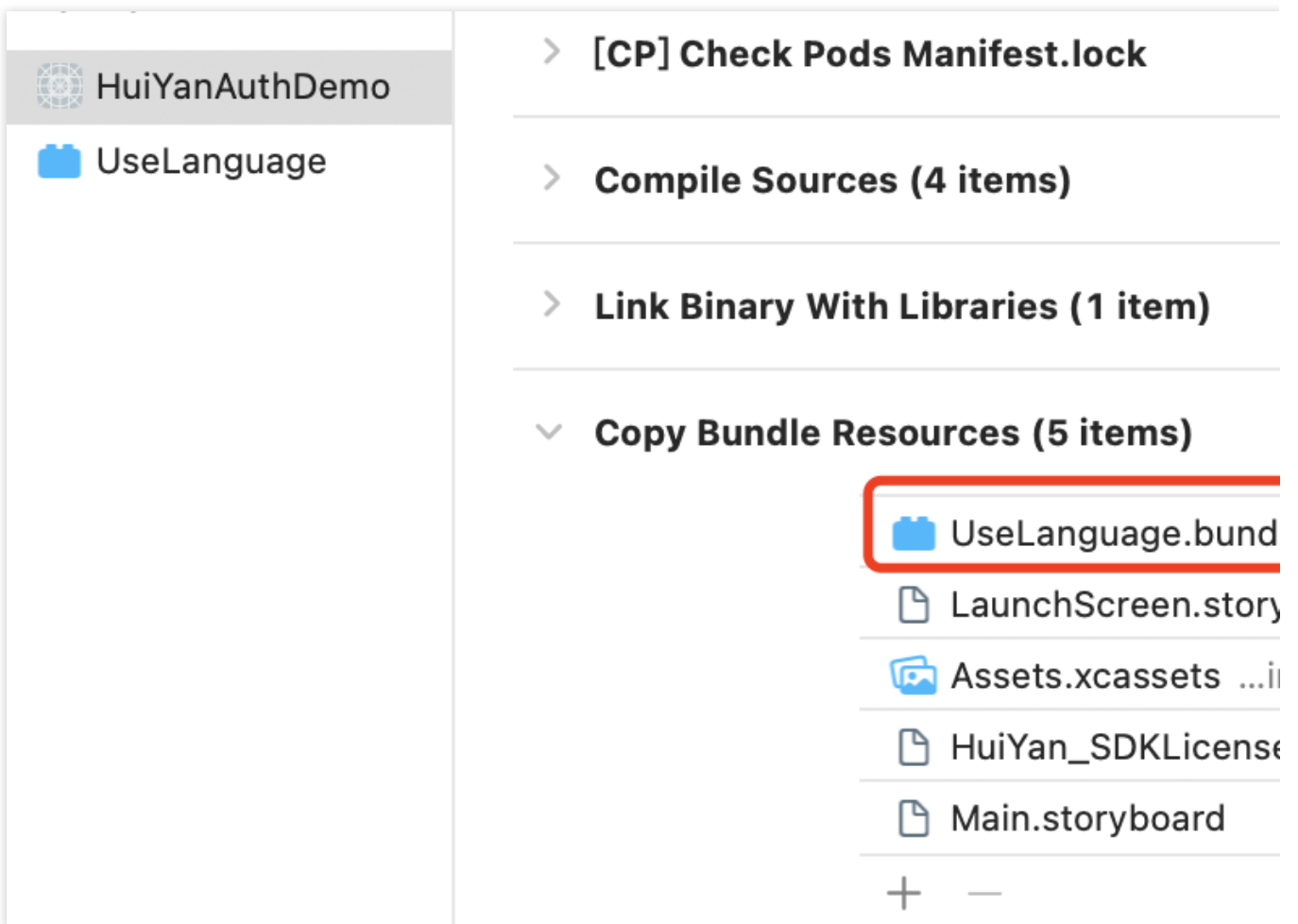
```
customerConfig.userLanguageBundleName = @"UserUIBundle";  
customerConfig.userLanguageFileName = @"en.lproj";
```

`userLanguageFileName` can be used to view the corresponding filename in the compiled `bundle` file.

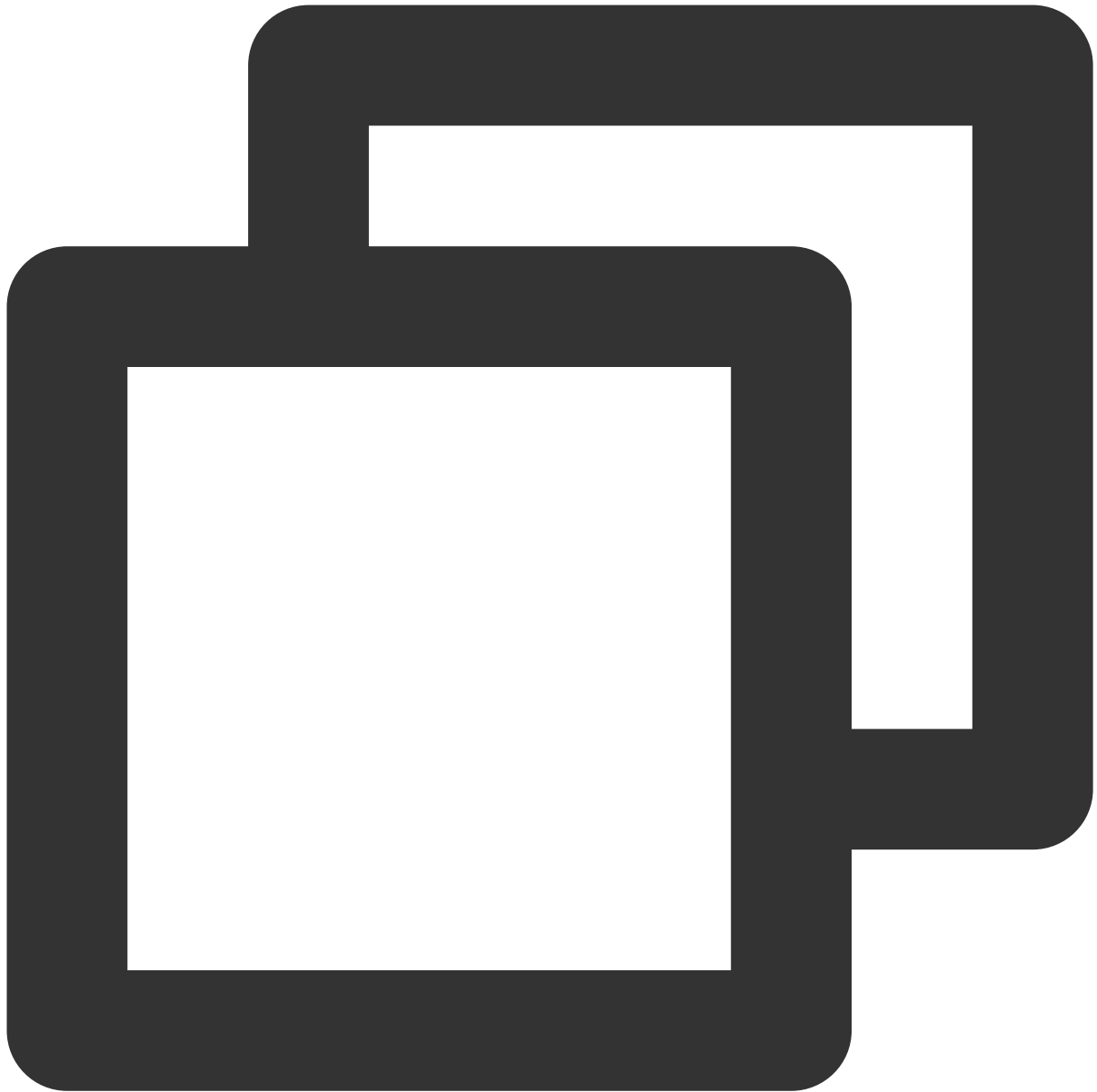


Setting a custom SDK language

1. Add the custom `UseLanguage.bundle` to the project (Copy Bundle Resources).



2. Configure the settings as follows:



```
HuiYanOsConfig *config = [[HuiYanOsConfig alloc] init];  
config.languageType = CUSTOMIZE_LANGUAGE;  
config.userLanguageFileName = @"ko";//For example, set `ko.lproj`  
config.userLanguageBundleName = @"UseLanguage";//Custom bundle name, such as `UseLa
```

If `config.languageType = DEFAULT;` is set, the system will find the language file for the current region in the custom bundle, and if it cannot be found, the language will be `en` by default.

Maintenance method

Take the Demo project as a custom UI project. Modify the `bundle` source file in the Demo project, and build `bundle` to access your project. You need to maintain the Demo project by yourself.

FAQs

Last updated : 2023-05-09 11:22:11

This document describes FAQs related to the liveness detection and face comparison SDK and provides solutions.

Client

Android

1. What should I do if I receive the **Invoke-customs are only supported starting with Android O (--min-api 26)** error after integrating FaceID?

Add the following configuration to the `build.gradle` file:



```
// Java 1.8 is supported
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

2. If the integrator uses the obfuscation tool AndResGuard, you can add the following obfuscation configuration:



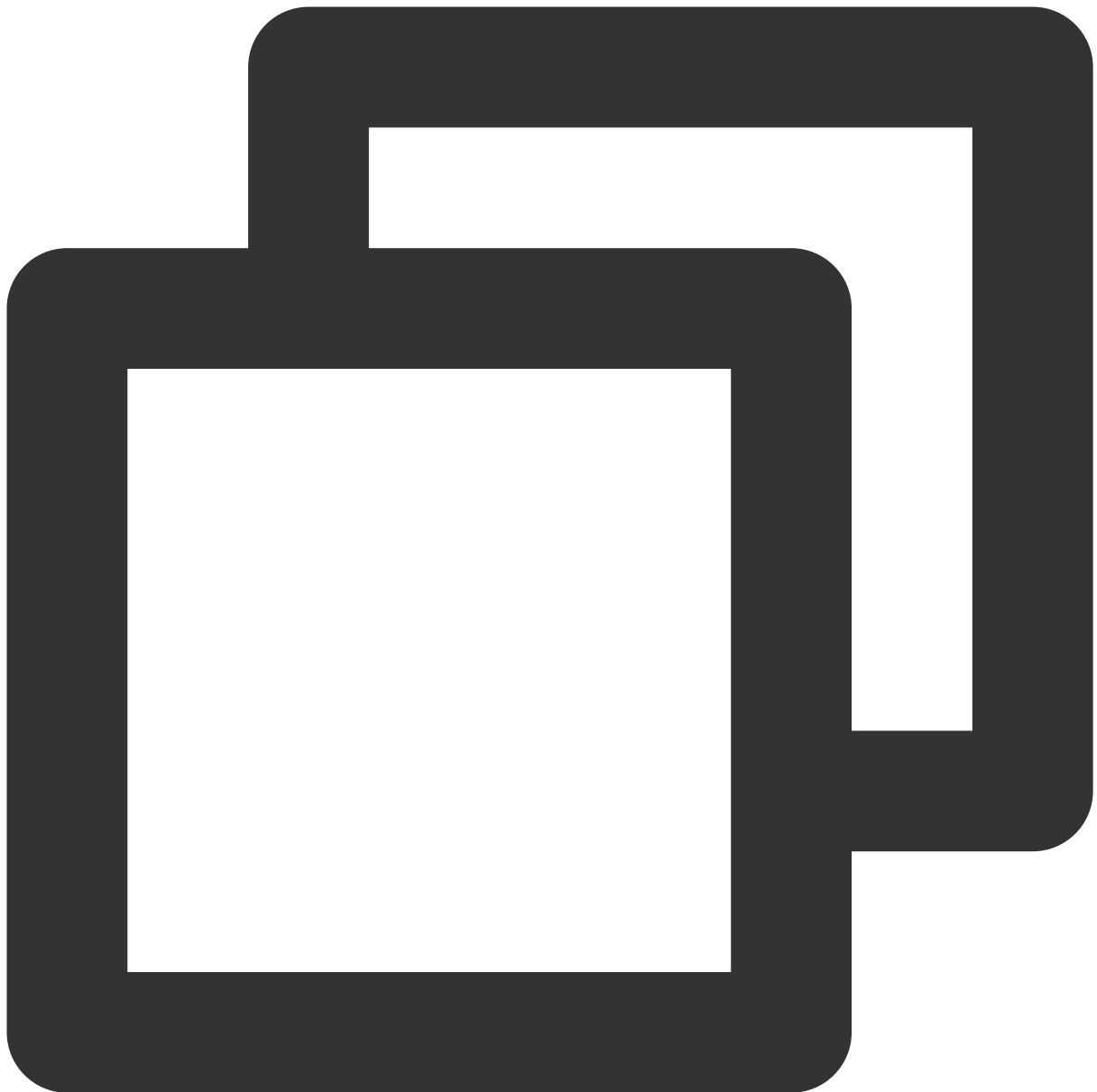
```
// for HuiYanSDK
"R.string.ocr_",
"R.string.rst_",
"R.string.net_",
"R.string.msg_",
"R.string.fl_",
```

3. If Android X reports **android.content.res.Resources\$NotFoundException:from xml type xml resource ID #0x7f0800c3** on devices with an earlier version of system, you can add dependent vector diagrams.



```
// Vector diagrams for earlier versions  
implementation 'androidx.vectordrawable:vectordrawable:1.1.0'
```

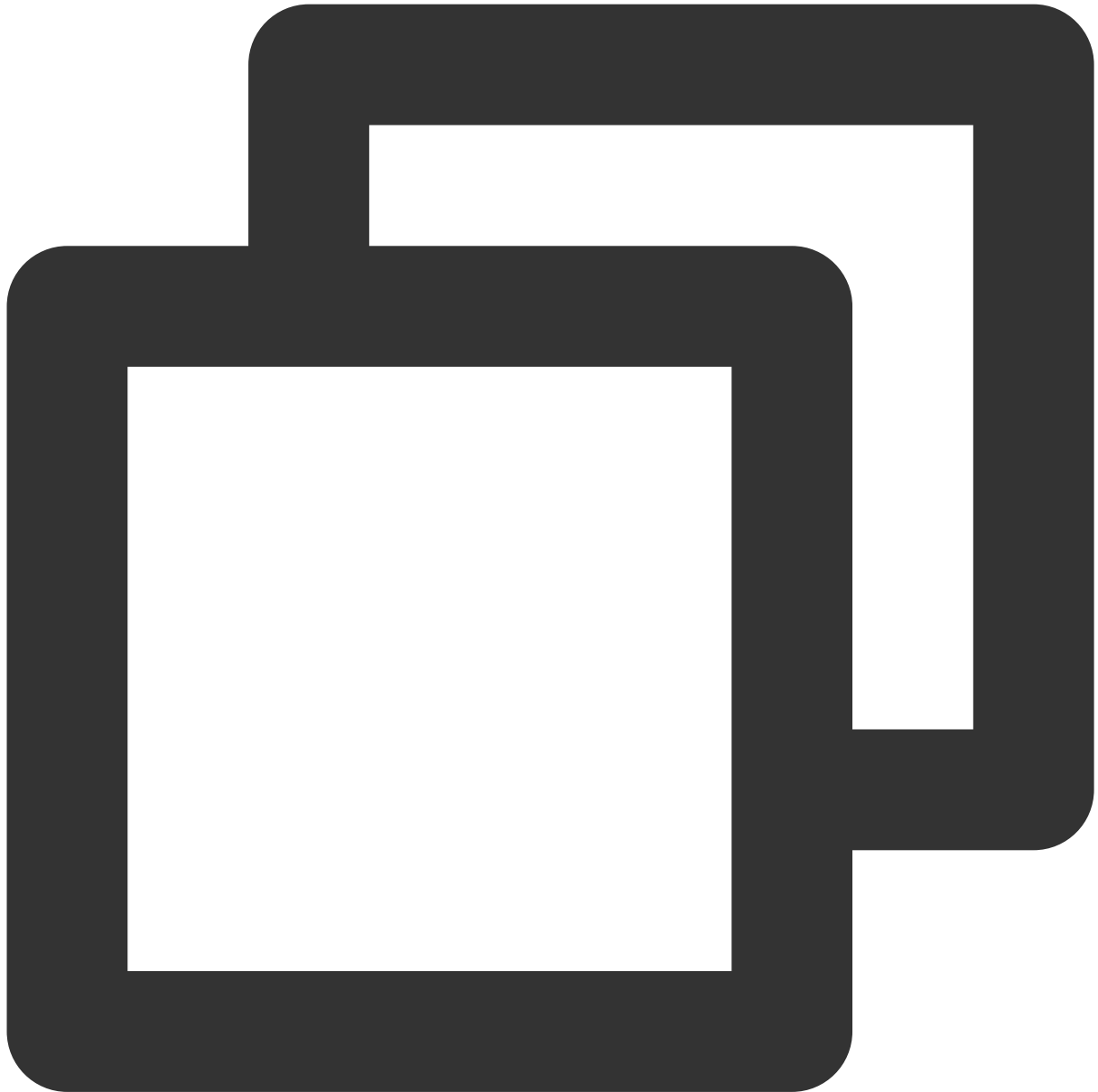
4. If **Android Support** reports the following errors on devices with an earlier version (v6.0 or earlier) of system:



```
android.content.res.Resources$NotFoundException: File res/drawable/$txy_face_id_log
    at android.content.res.Resources.loadColorStateListForCookie (Resources.java:2749)
    at android.content.res.Resources.loadColorStateList (Resources.java:2749)
    at android.content.res.TypedArray.getColor (TypedArray.java:441)
    at android.content.res.XResources$XTypedArray.getColor (XResources.java:128)
    at android.support.v4.content.res.TypedArrayUtils.getNamedColor (TypedArrayUtils.java:128)
    at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.update (VectorDrawableCompat$VFullPath.java:128)
    at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.inflate (VectorDrawableCompat$VFullPath.java:128)
    at android.support.graphics.drawable.VectorDrawableCompat.inflateInternal (VectorDrawableCompat.java:128)
    at android.support.graphics.drawable.VectorDrawableCompat.inflate (VectorDrawableCompat.java:128)
    at android.support.graphics.drawable.VectorDrawableCompat.createFromXmlInner (VectorDrawableCompat.java:128)
```

```
at android.support.v7.widget.AppCompatDrawableManager$VdcInflateDelegate.c
```

You need to update the support dependencies to the latest version (v28.0.0):



```
implementation 'com.android.support:appcompat-v7:28.0.0'  
// A component library compatible with vector diagrams for earlier versions  
implementation 'com.android.support:support-vector-drawable:28.0.0'  
implementation 'com.android.support:animated-vector-drawable:28.0.0'
```

iOS

1. What should I do if the SDK crashes and the following log is printed when I enter the SDK: "**reason: '** - [__NSDictionaryM setObject:forKey:]: key cannot be nil'**"? Solution: Add **-ObjC** in **Build Settings > Other Linker Flags**.
2. What should I do if the following information is displayed during compilation:
Undefined symbol: _vImageConvert_Planar16FtoPlanarF
Undefined symbol: _vImageConvert_PlanarFtoPlanar16F
Solution: Add the system library `Accelerate.framework`.
3. What should I do if "**face-tracker-v001 bundle path is nil**" or "**HuiYanSDKUI bundle path is nil**" is prompted?
Solution: Add the two prompted resource files to `Copy Bundle Resources`.

Integrating Identity Verification (App SDK)

Integration Process

Last updated : 2024-01-19 14:57:50

This document describes the process of integrating the Identity Verification (App SDK).

Preparations

Sign up for a Tencent Cloud account. For more information, see [Signing Up](#).

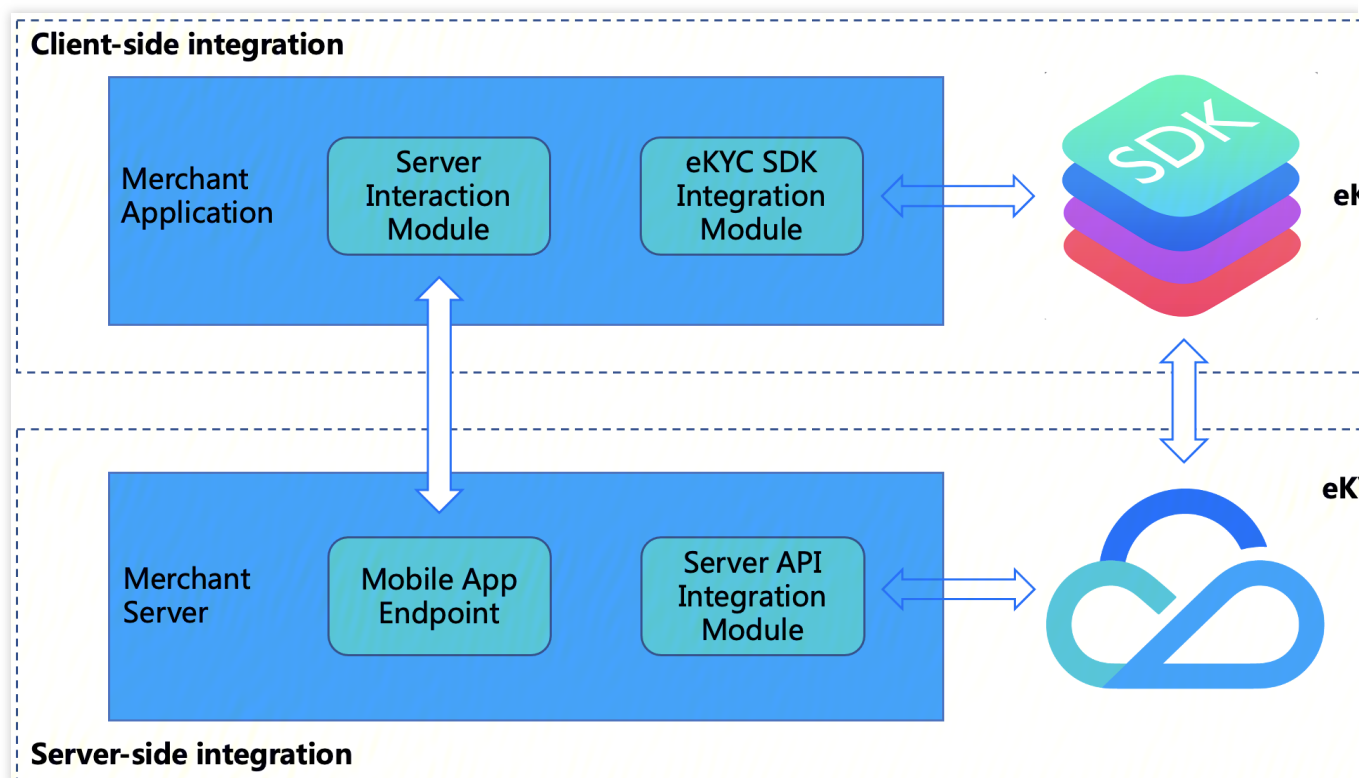
Complete enterprise identity verification. For more information, see [Enterprise Identity Verification Guide](#).

Log in to the [eKYC console](#) and activate the service.

[Contact us](#) to obtain the latest SDK and license.

Overall Architecture

The following figure shows the architecture for integrating the Identity Verification (App SDK) of Tencent Cloud eKYC.



The SDK integration includes two parts:

A. Client-side integration: Integrate the SDK into the customer's client app.

B. Server-side integration: Expose the endpoint of your (merchant) application to your (merchant) server so that the merchant application can interact with the merchant server and then access the server API to obtain `SdkToken`, which is used throughout the liveness detection and face comparison process and to pull the final verification result.

Overall Interaction Process

The integrator only needs to pass in the token and start the Identity Verification (App SDK) to achieve user identity verification, which includes ID document OCR, liveness detection, and face comparison. After the end user completes the verification, the integrator can obtain the verification result using an API.

API for getting the token: [ApplySdkVerificationToken](#)

API for pulling the identity verification result: [GetSdkVerificationResult](#)

The following diagram shows the overall logic of interaction between the following modules:

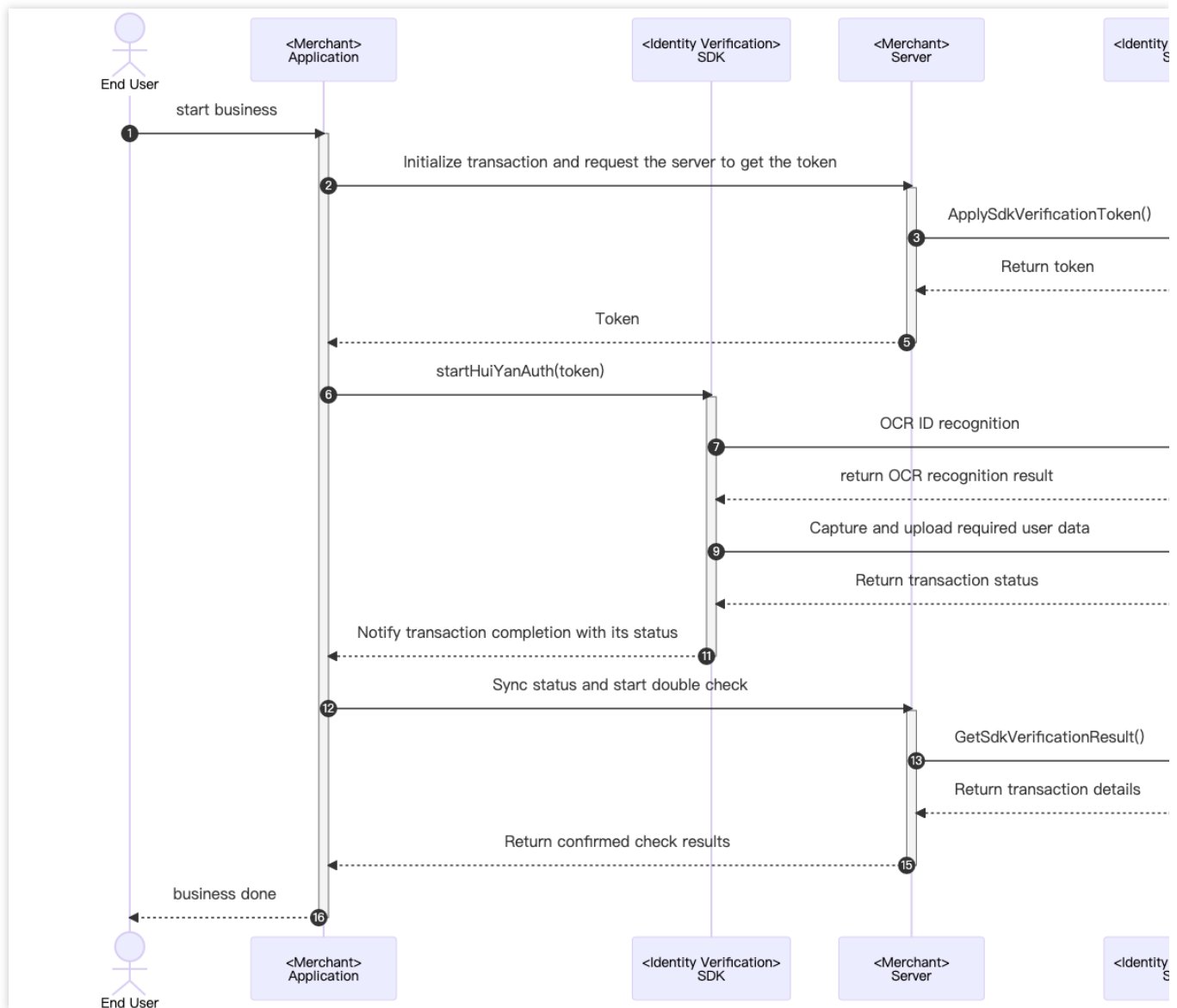
End User

Identity Verification (App SDK): The SDK obtained during the preparation stage.

Merchant Application: The customer business app that uses and is integrated with the Identity Verification (App SDK).

Merchant Server: The customer server.

Identity Verification Server: Tencent Cloud Identity Verification backend service API.



The recommended detailed interaction process is as follows:

1. An end user triggers the merchant application on the terminal to call the identity verification service scenario.
2. The merchant application sends a request to the merchant server to notify that the liveness detection service token is required for starting identity verification once.
3. The merchant server passes in relevant parameters to call the TencentCloud API [ApplySdkVerificationToken](#).
4. After receiving the request for calling [ApplySdkVerificationToken](#), the identity verification server delivers the token to the merchant server.
5. The merchant server delivers the obtained service token to the customer's merchant application.
6. The merchant application calls the Identity Verification (App SDK)'s startup API **startHuiYanAuth** to pass in the token and configuration information and starts identity verification.
7. The Identity Verification (App SDK) starts OCR by uploading the document photo to the identity verification server to recognize the user's document information.
8. The identity verification server returns the result to the Identity Verification (App SDK).

9. The Identity Verification (App SDK) captures and uploads the required user data, including liveness data, to the identity verification server.
10. After completing liveness detection and face comparison, the identity verification server returns the result to the Identity Verification (App SDK).
11. The Identity Verification (App SDK) actively triggers callback to notify the merchant application that the verification is complete and of the verification status.
12. After receiving the callback, the merchant application sends a request to notify the merchant server to obtain the verification result for confirmation.
13. The merchant server actively calls the identity verification server API [GetSdkVerificationResult](#) with relevant parameters and the service token passed in to obtain the verification result.
14. After receiving the request for calling [GetSdkVerificationResult](#), the identity verification server returns the verification result to the merchant server.
15. After receiving the verification result, the merchant server delivers the required information to the merchant application.
16. The merchant application displays the final result on the UI to notify the user of the verification result.

Integration Process

Server-side integration

1. Preparations

Before server-side integration, you need to activate the Tencent Cloud eKYC service and obtain TencentCloud API access key `SecretId` and `SecretKey` by following the instructions in [Getting API Key](#). In addition, you need to follow the instructions in [Connecting to TencentCloud API]

(<https://www.tencentcloud.com/document/product/1061/54960!398ac9c1781426f85199f8704c2ae268>) to import the SDK package with the programming language you are familiar with to your server modules, to ensure that the TencentCloud API can be successfully called and API requests and responses can be properly processed.

2. Integration

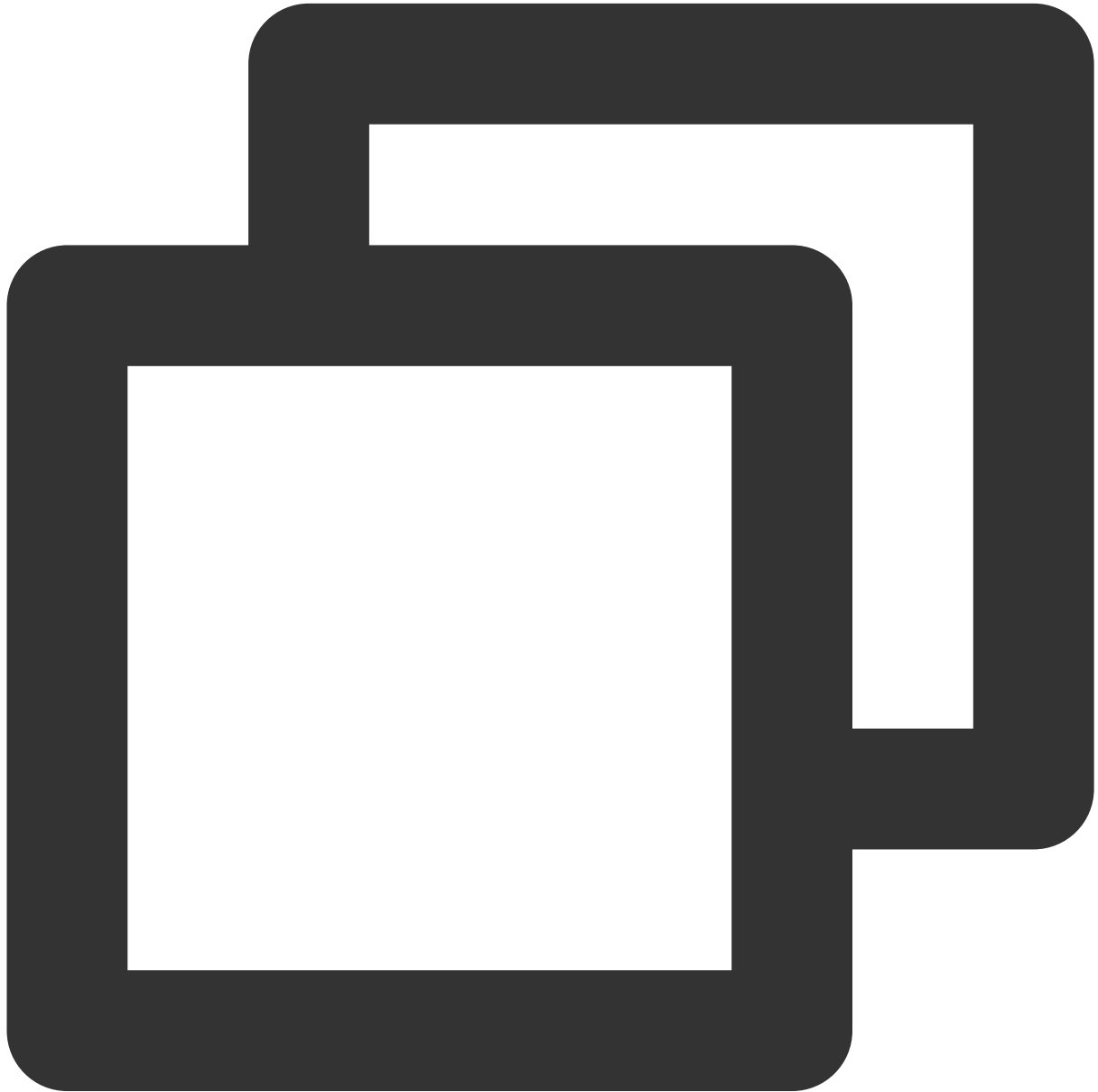
To ensure that your (merchant) client application interacts with your (merchant) server, the merchant server needs to call the API [ApplySdkVerificationToken](#) provided by eKYC to obtain `SDKToken`, which is used throughout the identity verification process and used by the API [\[GetSdkVerificationResult\]](#)

(<https://www.tencentcloud.com/document/product/1061/49951!f60fbe65c5f7cb4584259d37c8176b67> to obtain the verification result. The merchant server also needs to provide the corresponding endpoint for the merchant client to call. The following sample code with the Golang language is used as an example to show how to call TencentCloud API on the server and obtain the correct response.

Note: This example only demonstrates the processing logic required for interaction between the merchant server and TencentCloud API service. If necessary, you need to implement your own business logic, for example:

After you obtain the `SDKToken` using the **ApplySdkVerificationToken** API, you can return other responses required by the client application to the client along with the `SDKToken`.

After you obtain the identity verification result using the **GetSdkVerificationResult** API, you can save the returned photo with the best frame rate for later business logic.



```
var FaceIdClient *faceid.Client

func init() {
```

```

    // Instantiate a client configuration object. You can specify the timeout p
    prof := profile.NewClientProfile()
    prof.HttpProfile.ReqTimeout = 60
    // TODO replace the SecretId and SecretKey string with the API SecretId and
    credential := cloud.NewCredential("SecretId", "SecretKey")
    var err error
    // Instantiate the client object of the requested faceid
    FaceIdClient, err = faceid.NewClient(credential, "ap-singapore", prof)
    if nil != err {
        log.Fatal("FaceIdClient init error: ", err)
    }
}

// ApplySdkVerificationToken get token
func ApplySdkVerificationToken(w http.ResponseWriter, r *http.Request) {
    log.Println("get face id token")
    // Step 1: ... parse parameters
    _ = r.ParseForm()
    var IdCardType = r.FormValue("IdCardType")
    var NeedVerifyIdCard = false

    // Step 2: instantiate the request object and provide necessary parameters
    request := faceid.NewApplySdkVerificationTokenRequest()
    request.IdCardType = &IdCardType
    request.NeedVerifyIdCard = &NeedVerifyIdCard
    // Step 3: call the Tencent Cloud API through FaceIdClient
    response, err := FaceIdClient.ApplySdkVerificationToken(request)

    // Step 4: process the Tencent Cloud API response and construct the return
    if nil != err {
        log.Println("error: ", err)
        _, _ = w.Write([]byte("error"))
        return
    }
    SdkToken := response.Response.SdkToken
    apiResp := struct {
        SdkToken *string
    }{SdkToken: SdkToken}
    b, _ := json.Marshal(apiResp)

    // ... more codes are omitted

    //Step 5: return the service response
    _, _ = w.Write(b)
}

// GetSdkVerificationResult get result

```

```
func GetSdkVerificationResult(w http.ResponseWriter, r *http.Request) {
    // Step 1: ... parse parameters
    _ = r.ParseForm()
    SdkToken := r.FormValue("SdkToken")
    // Step 2: instantiate the request object and provide necessary parameters
    request := faceid.NewGetSdkVerificationResultRequest()
    request.SdkToken = &SdkToken
    // Step 3: call the Tencent Cloud API through FaceIdClient
    response, err := FaceIdClient.GetSdkVerificationResult(request)

    // Step 4: process the Tencent Cloud API response and construct the return
    if nil != err {
        _, _ = w.Write([]byte("error"))
        return
    }
    result := response.Response.Result
    apiResp := struct {
        Result *string
    }{Result: result}
    b, _ := json.Marshal(apiResp)

    // ... more codes are omitted

    //Step 5: return the service response
    _, _ = w.Write(b)
}

func main() {
    // expose endpoints
    http.HandleFunc("/api/v1/get-token", ApplySdkVerificationToken)
    http.HandleFunc("/api/v1/get-result", GetSdkVerificationResult)
    // listening port
    err := http.ListenAndServe(":8080", nil)
    if nil != err {
        log.Fatal("ListenAndServe error: ", err)
    }
}
```

Note: For the full code example, see [faceid-server-demo](#).

3. API testing

After you complete the integration, you can test whether the current integration is correct by running the postman or curl command. To be specific, access the API (<http://ip:port/api/v1/get-token>) to check whether `SdkToken` is returned and access the API (<http://ip:port/api/v1/get-result>) to check whether the value of the `Result` field is 0.

Through these results, you can determine whether the server-side integration is successful. For details on responses, see API description.

Integration with Android

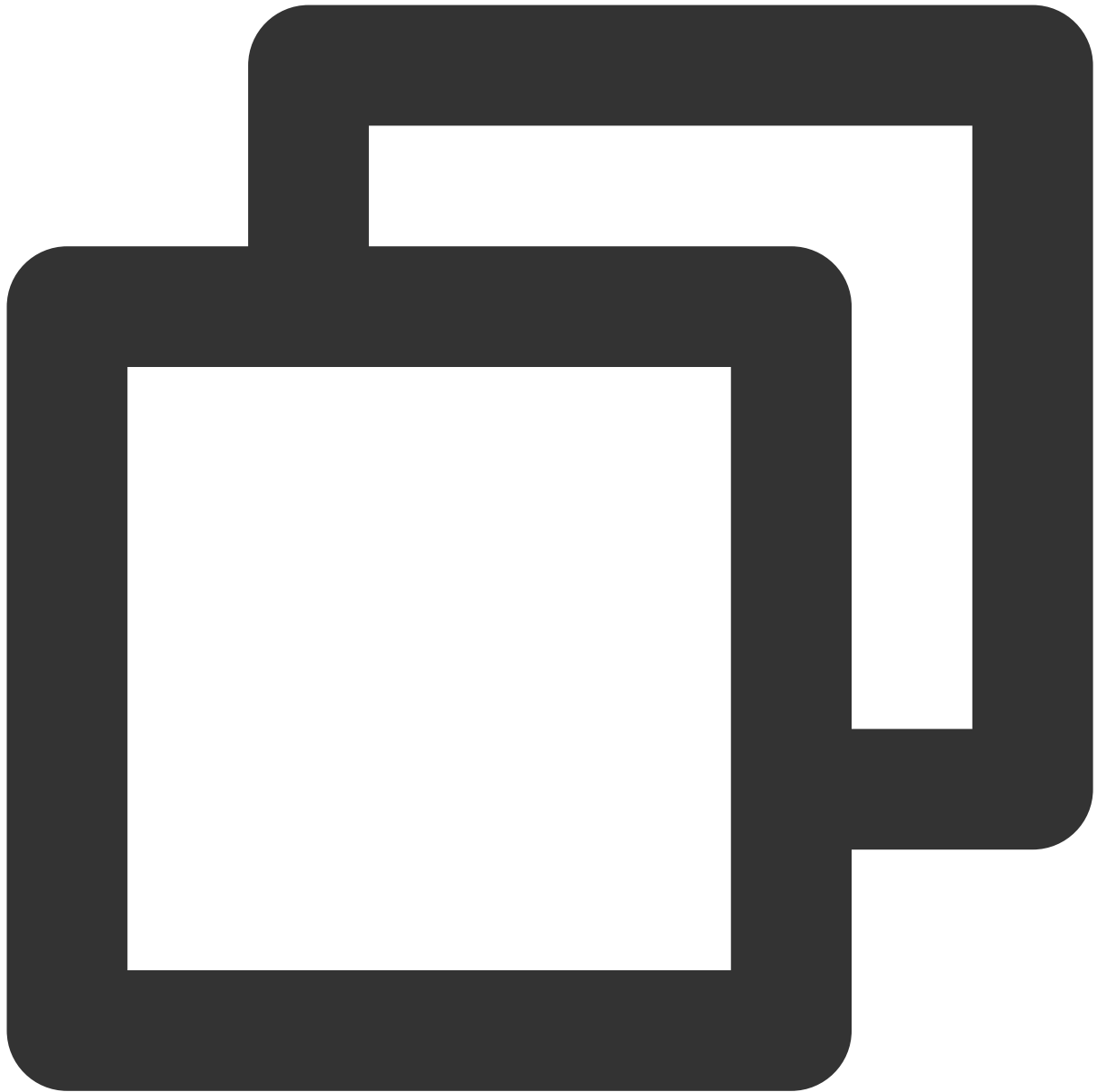
1. Dependent environment

The current SDK for Android is applicable to API 19 (Android 4.4) and later.

2. SDK integration steps

1. Add the following files to the **libs** directory of your project: **ekyc_android_1.0.x.x_release.aar**, **huiyansdk_android_1.0.x.x_release.aar**, **OcrSDK-private-v1.0.x.x-release.aar**, **OcrSDK-common-model-v1.0.x.x-release.aar**, **tencent-ai-sdk-youtu-base-v1.0.x.x-release.aar**, **tencent-ai-sdk-common-v1.0.x.x-release.aar**, and **tencent-ai-sdk-aicamera-v1.0.x.x-release.aar** (the version numbers of the files downloaded from the official website apply).

2. Configure **build.gradle** in your project as follows:

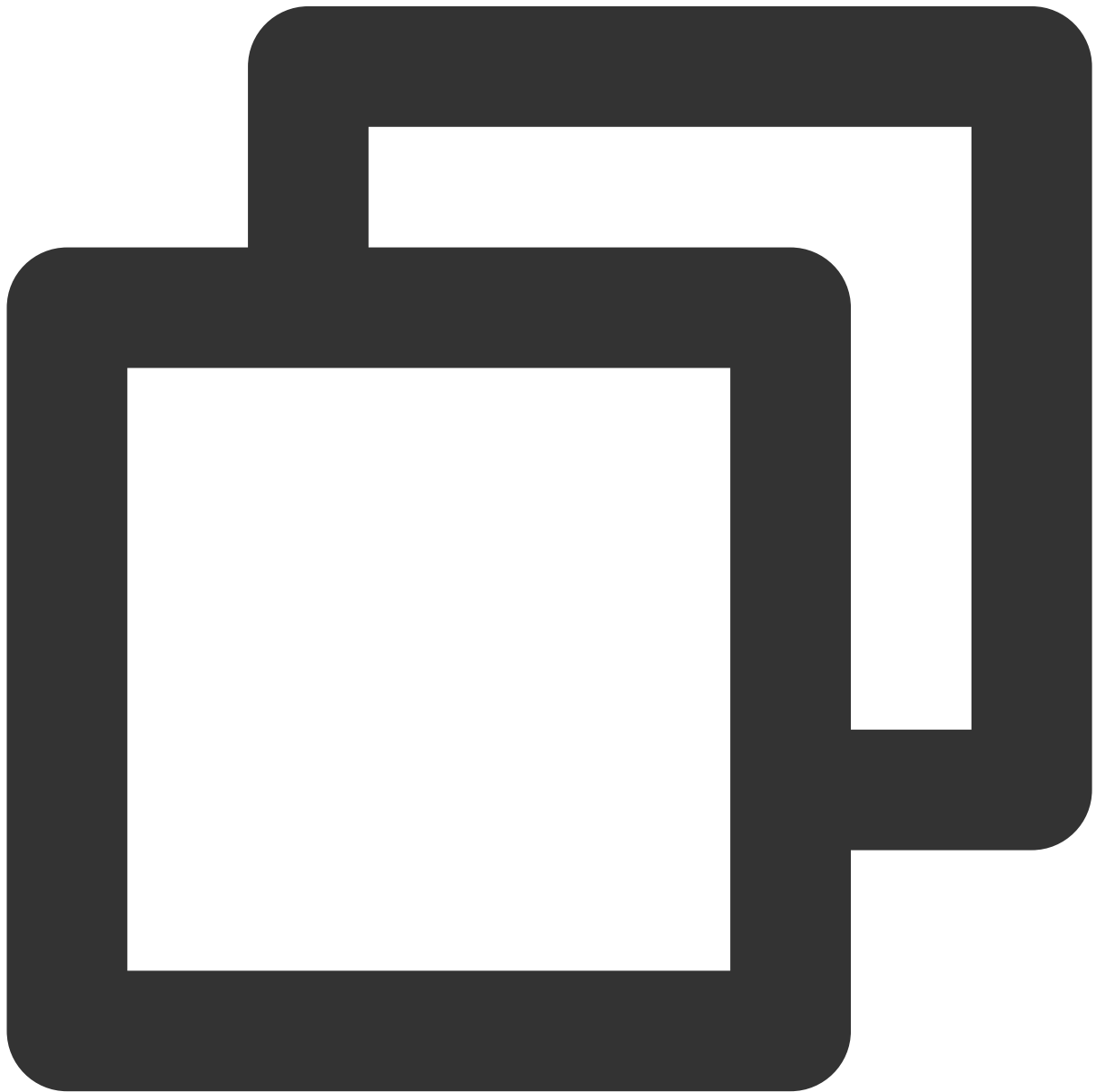


```
// Set .so architecture filtering in NDK (using armeabi-v7a as an example)
ndk {
    abiFilters 'armeabi-v7a'
}

dependencies {
    // Identity Verification (App SDK) version
    implementation files("libs/ekyc_android_1.0.x.x_release.aar")
    // Identity verification components
    implementation files("libs/huiyansdk_android_1.0.x.x_release.aar")
    // OCR components
```

```
implementation files("libs/OcrSDK-common-model-v1.x.x-release.aar")
implementation files("libs/OcrSDK-private-v2.x.x-release.aar")
// Common capability components
implementation files("libs/tencent-ai-sdk-youtu-base-1.0.x.x-release.aar")
implementation files("libs/tencent-ai-sdk-common-1.x.x-release.aar")
implementation files("libs/tencent-ai-sdk-aicamera-1.x.x-release.aar")
// Import a `gson` library
implementation 'com.google.code.gson:gson:2.8.5'
}
```

3. You also need to make the necessary permission declaration in the `AndroidManifest.xml` file.



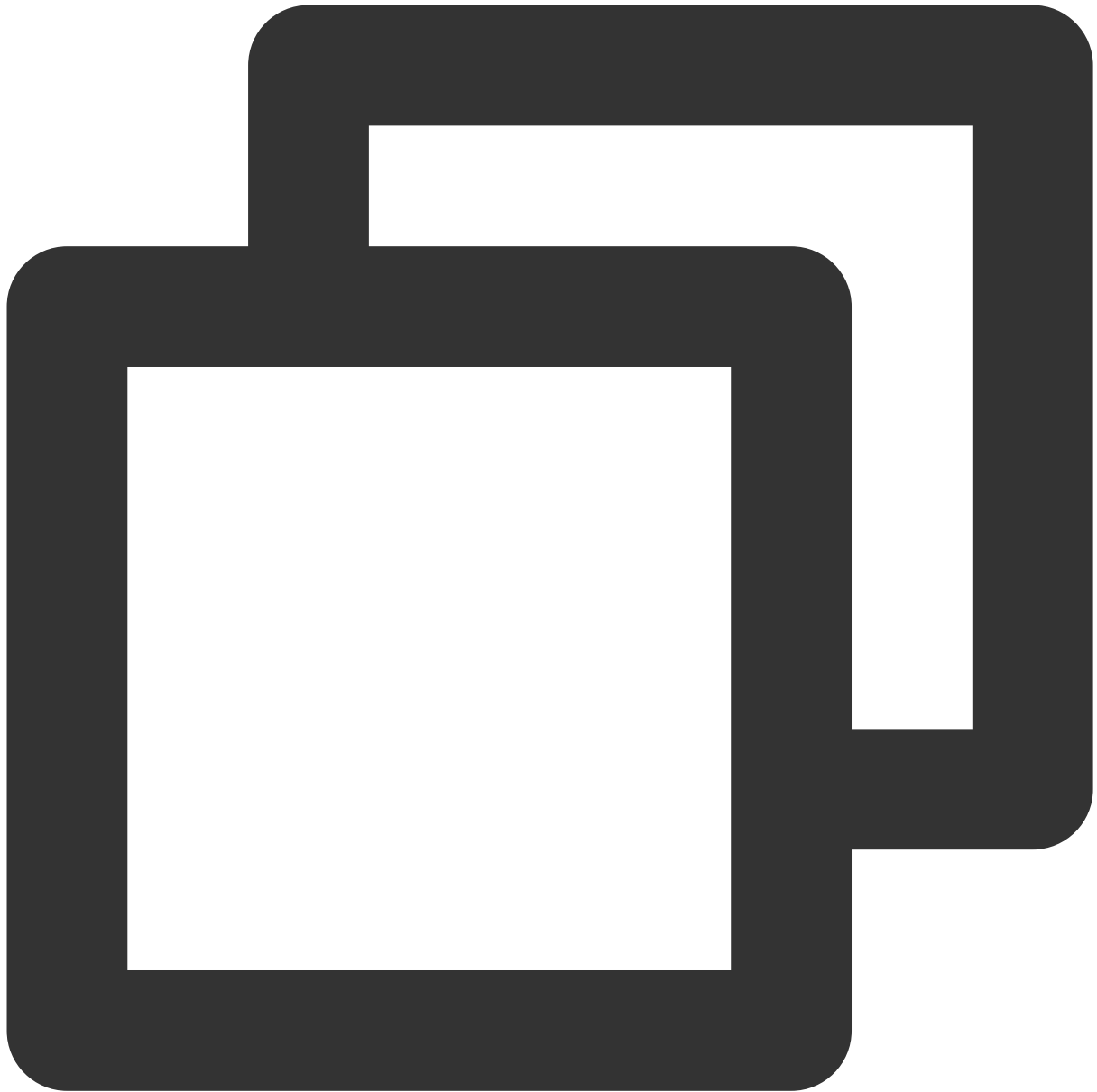

```
<!-- Camera permission -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />
<uses-feature android:name="android.hardware.camera.autofocus" />

<!-- Permissions required by the SDK -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<!-- Permissions optional for the SDK -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

If your app needs to be compatible with Android 6.0 or later, in addition to declaring the above permissions in the `AndroidManifest.xml` file, you need to add the code **Dynamically apply for permissions**.

3. API initialization

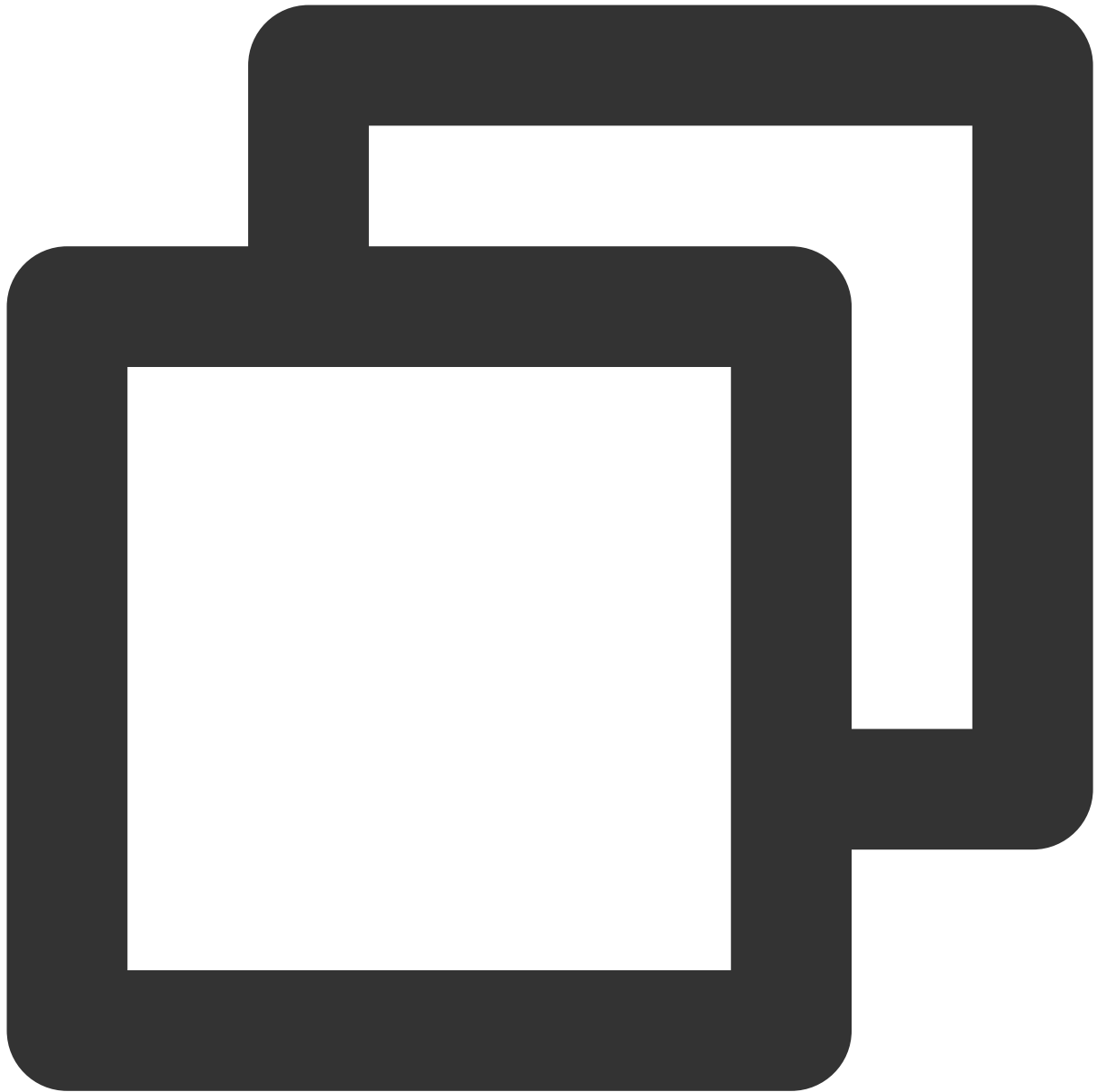
This API is called during app initialization, which is mainly used to perform some initialization operations for the SDK. We recommend you call this API in `Application`.



```
@Override
public void onCreate() {
    super.onCreate();
    EkycHySdk.init(this);
}
```

4. Starting the identity verification process

When you need to start the identity verification process, call the function `EkycHySdk.startEkycCheck()`, and pass in the `sdkToken` required for this process, configuration information, and the callback for listening for the result.



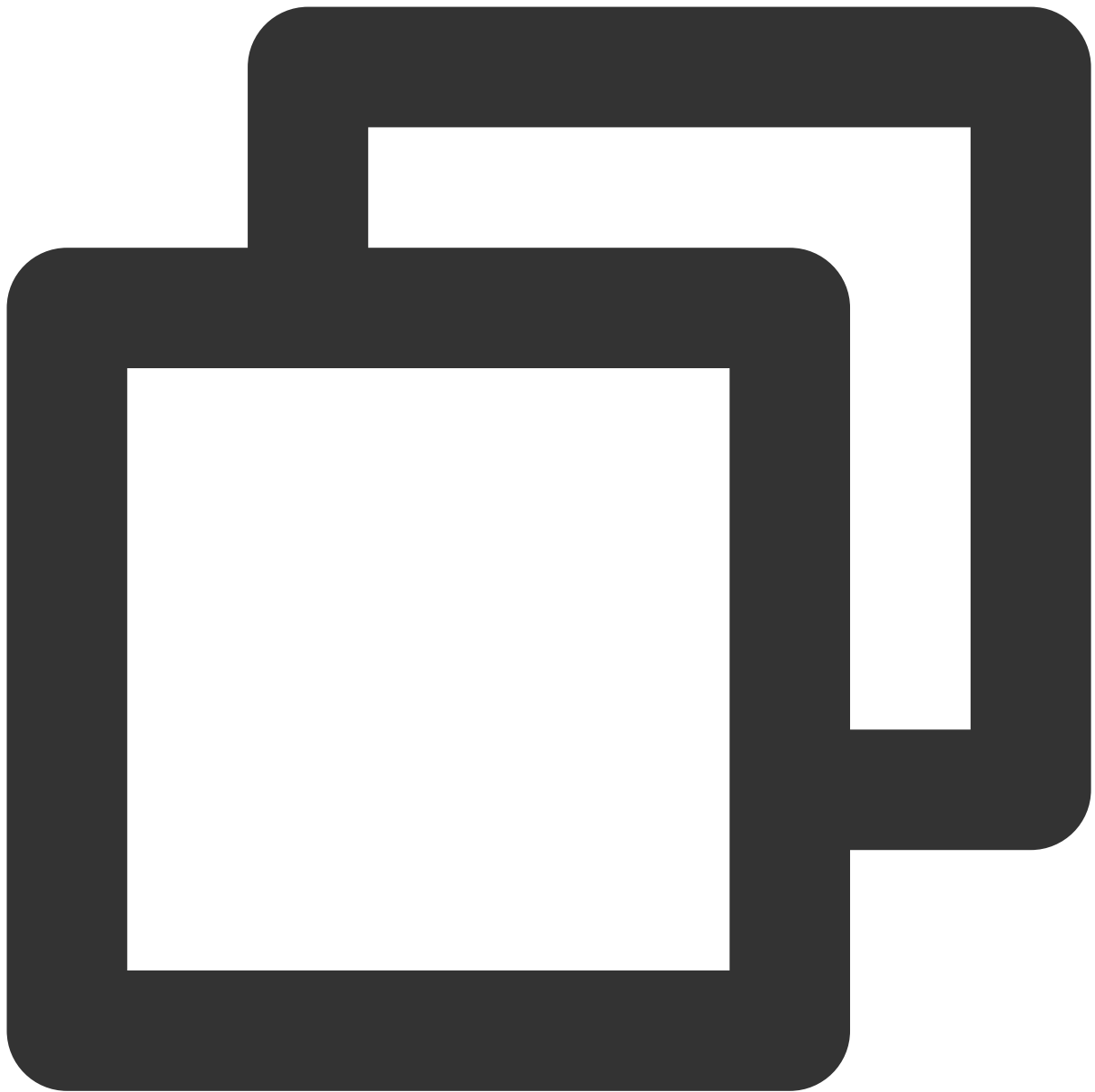
```
// Set startup configurations
EkycHyConfig ekycHyConfig = new EkycHyConfig();
// Set the license name
ekycHyConfig.setLicenseName("ekycLicense.license");
ekycHyConfig.setOcrType(OcrRegionType.HK);
// Customize UI configurations
OcrUiConfig config = new OcrUiConfig();
ekycHyConfig.setOcrUiConfig(config);
// Set specific startup verification logic
// `sdkToken` is the unique credential obtained from the server for this process
EkycHySdk.startEkycCheck(sdkToken, ekycHyConfig, new EkycHyCallBack() {
```

```
@Override
public void onSuccess(EkycHyResult result) {
    Log.e(TAG, "result: " + result.toString());
    showToast ("Identity verification succeeded:" + result.toString());
}

@Override
public void onFail(int errorCode, String errorMsg, String ekycToken) {
    Log.e(TAG, "code: " + errorCode + " msg: " + errorMsg + " token: " + ekycToken)
    showToast ("Identity verification failed:" + "code: " + errorCode + " msg: " +
    }
});
```

sdkToken is the unique credential obtained from the server for this identity verification process.

Note: **ekycLicense.license** is the license file obtained from the sales rep or customer service, and you need to place it in the `assets` folder.



```
└─ src
  └─ main
    ├── assets
    └── └─ ekycLicense.license
```

5. Releasing SDK resources

Before your app exits, you can call the SDK resource release API.



```
@Override
protected void onDestroy() {
    EkycHySdk.release();
    super.onDestroy();
}
```

6. Configuring obfuscation rules

If the obfuscation feature is enabled for your app, add the following part to your obfuscation file to ensure the normal use of the SDK.



```
# Objects to be obfuscated
-keep class com.google.gson.** {*; }
-keep class com.tencent.cloud.** {*; }
-keep class com.tencent.youtu.** {*; }
-keep class com.tencent.cloud.ocr.** {*; }
-keep class com.tencent.cloud.ekyc.** {*; }
```

Note: For the full code example for Android, see [Android demo](#).

Integration with iOS

1. Dependent environment

1. Xcode 12.0 or later is required. We recommend you use the latest version.
2. The SDK for iOS is only supported by iOS 11.0 or later.

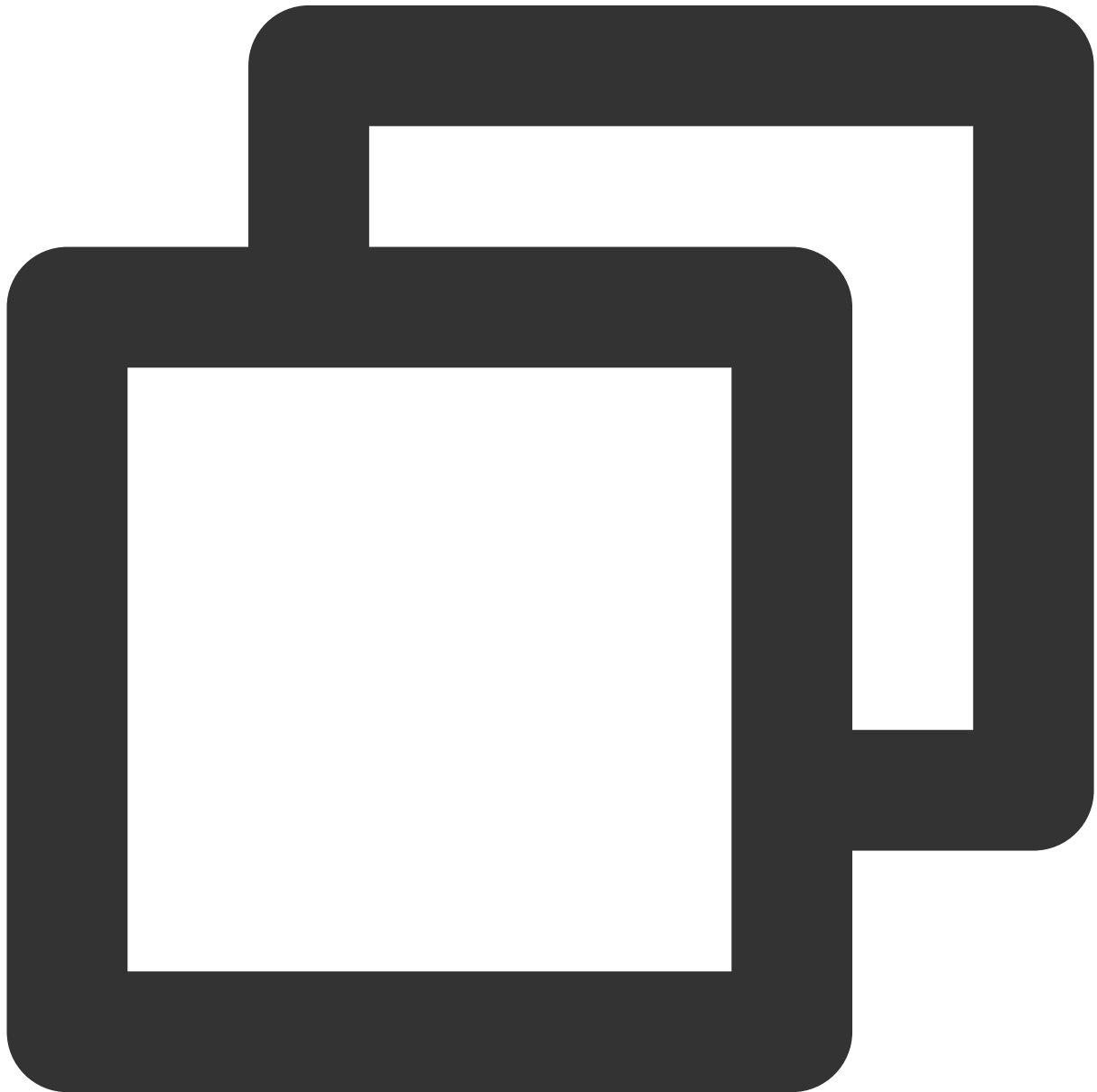
2. SDK integration steps

Manual integration

1. Import the relevant libraries and files.

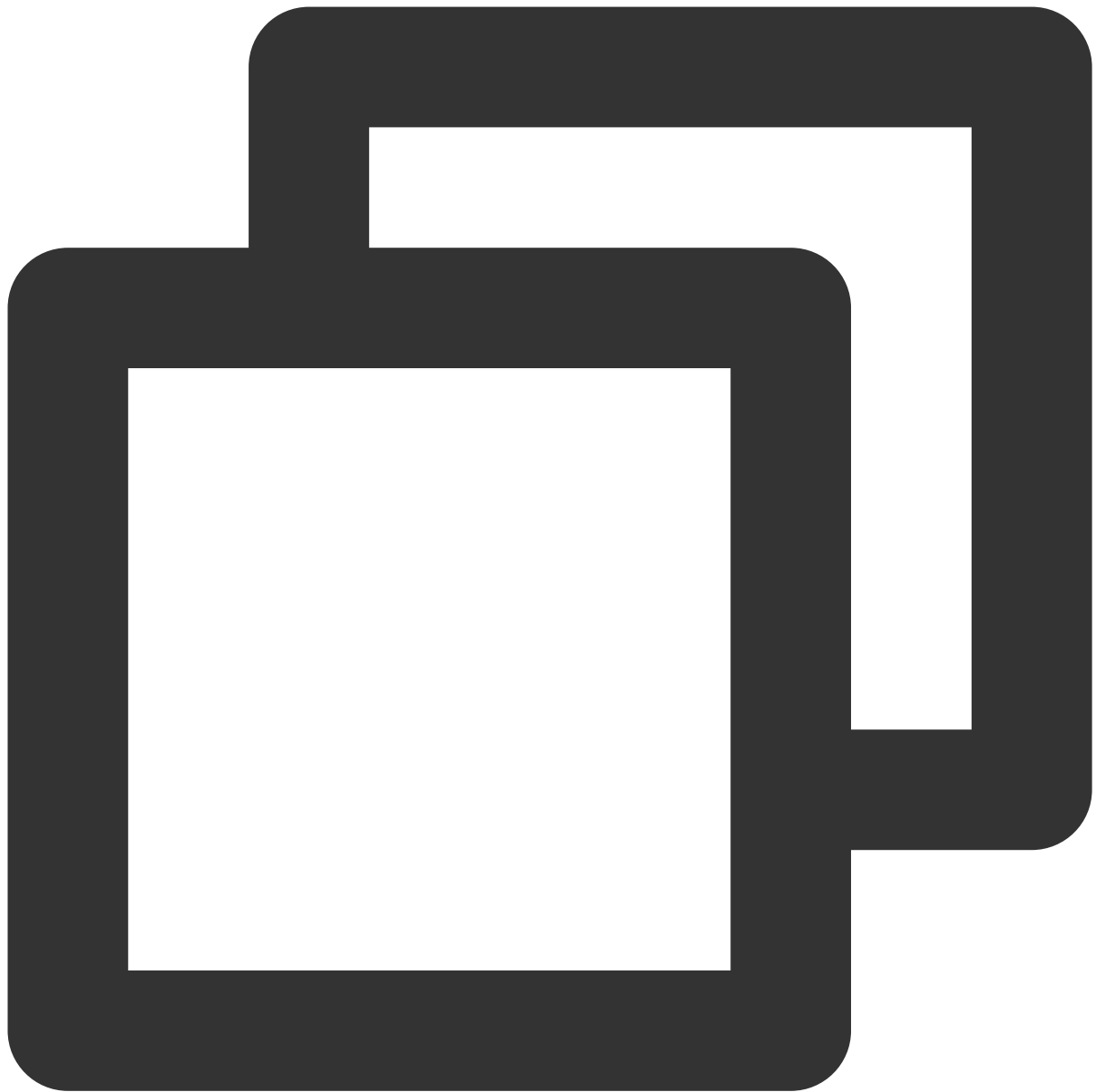
Import relevant frameworks in `Link Binary With Libraries` .

2. The SDK depends on the following libraries:



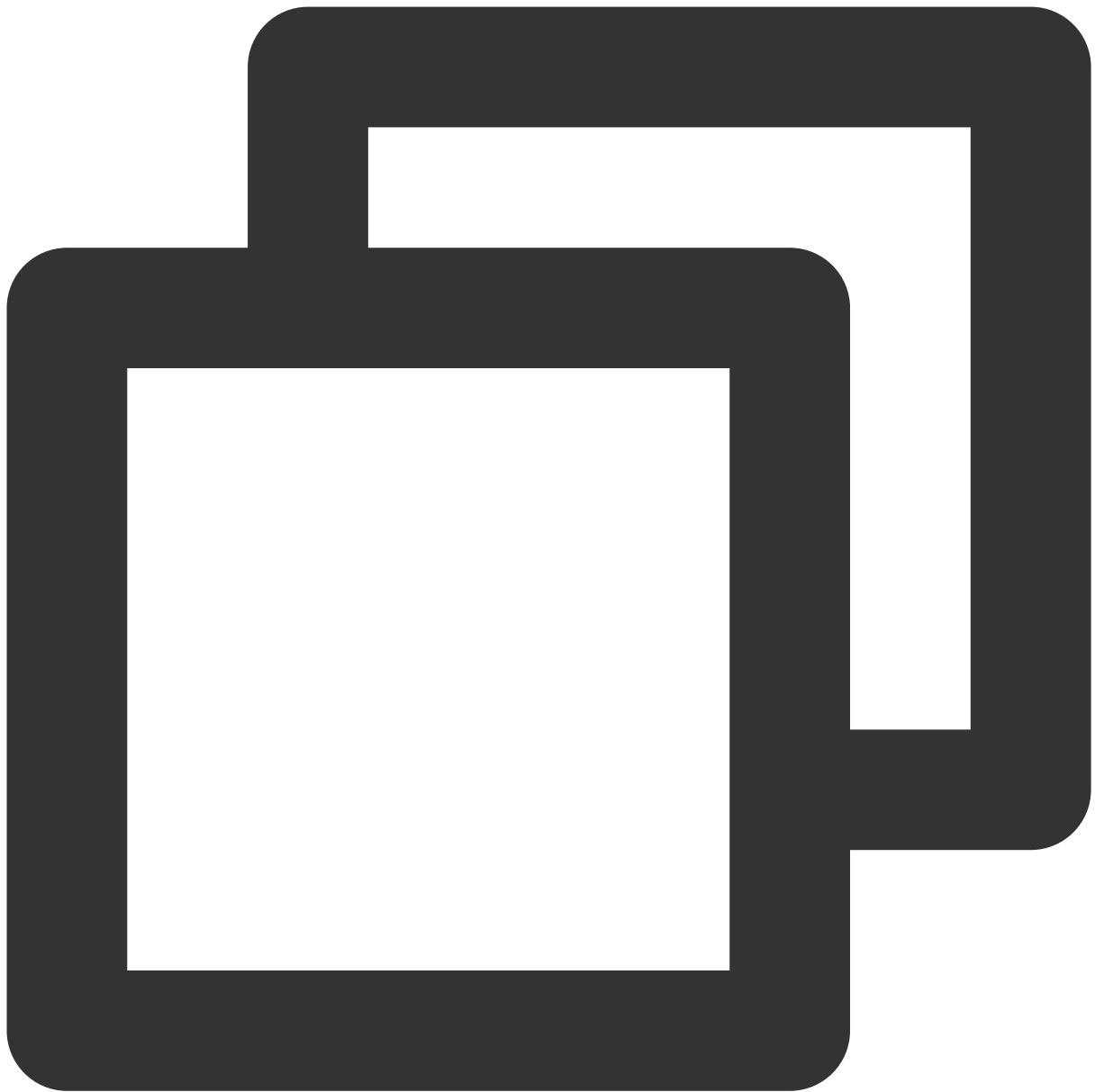

```
|— HuiYanEKYCVerification.framework
|— tnn.framework
|— tnnliveness.framework
|— YTCommonLiveness.framework
|— YTFaceAlignmentTinyLiveness.framework
|— YTFaceDetectorLiveness.framework
|— YTFaceLiveReflect.framework
|— YTFaceTrackerLiveness.framework
|— YTPoseDetector.framework
|— YtSDKKitActionLiveness.framework
|— YtSDKKitFramework.framework
|— YtSDKKitOcrVideoIdent.framework
|— YtSDKKitReflectLiveness.framework
|— YtSDKKitSilentLiveness.framework
|— HKOCRSDK.framework
|— tiny_opencv2.framework
|— HuiYanOverseasSDK.framework
|— IdVerification.framework
|— OcrSDKKit.framework
|— TXYCommonDevice.framework
|— TXYCommonNetworking.framework
|— TXYCommonUtils.framework
|— YTCv.framework
|— YTImageRefiner.framework
|— YtSDKKitFrameworkTool.framework
|— YTSm.framework
```

In `Link Binary With Libraries` , import the system framework.



```
├─ AVFoundation.framework
├─ libc++.tbd
├─ Accelerate.framework
└─ CoreML.framework
```

Import the resource file in `Copy Bundle Resources` .



```
|— face-tracker-v003.bundle
|— huiyan_verification.bundle
|— HuiYanSDKUI.bundle
|— idverificationres.bundle
|— ocr-v001.bundle
|— OcrSDK.bundle
|— ytsdkviidres.bundle
```

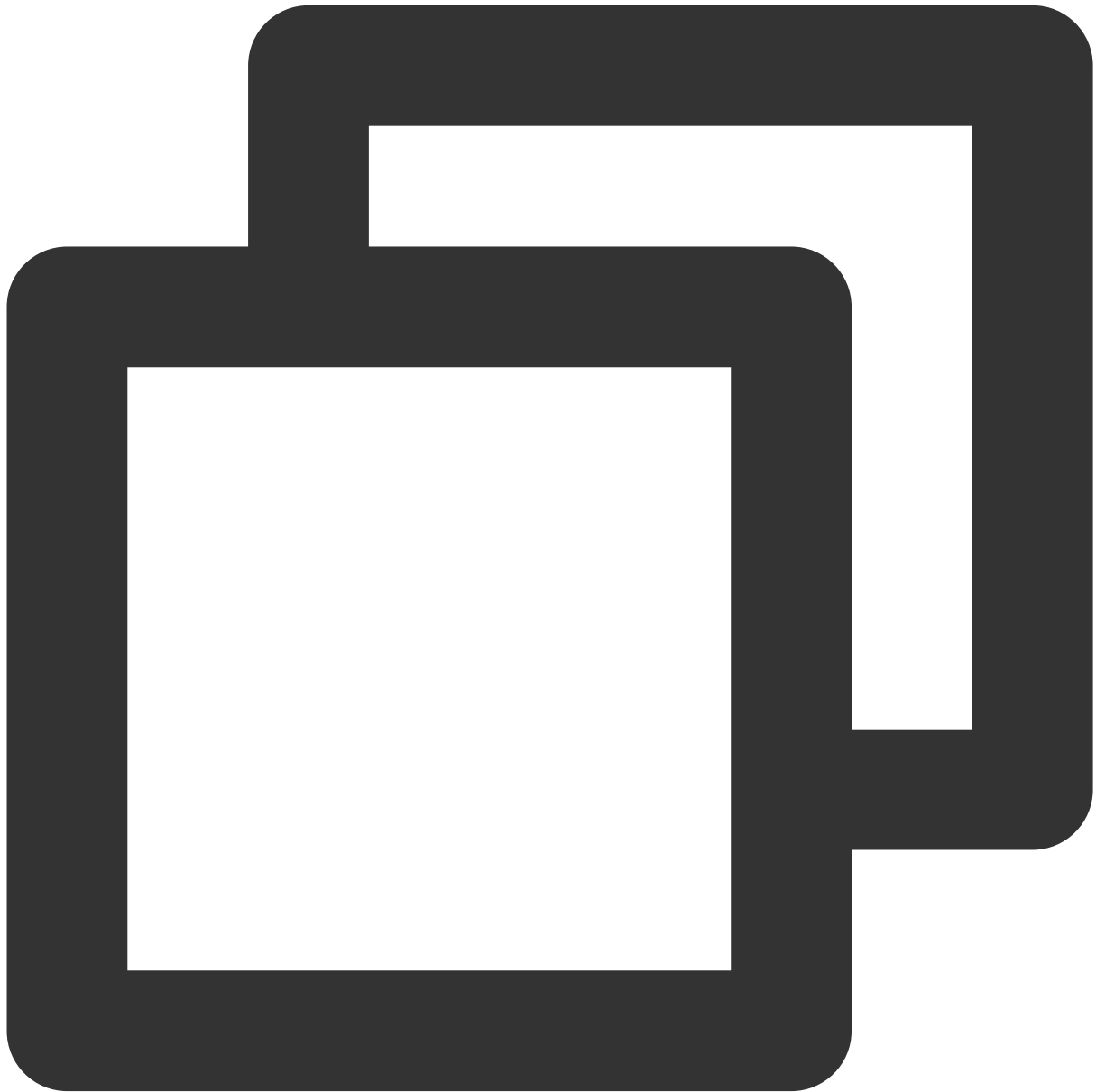
Build Phases settings

1. In `Other Linker Flags` , add `-ObjC` .

2. Integrate `ViewController.m` and set the extension to `.mm` (for a Swift project, add the system library `libc++.tbd`).

Permission settings

As the SDK requires a mobile network and camera permission, add the corresponding permission declarations and add the following `key-value` to the project's `info.plist` configuration.



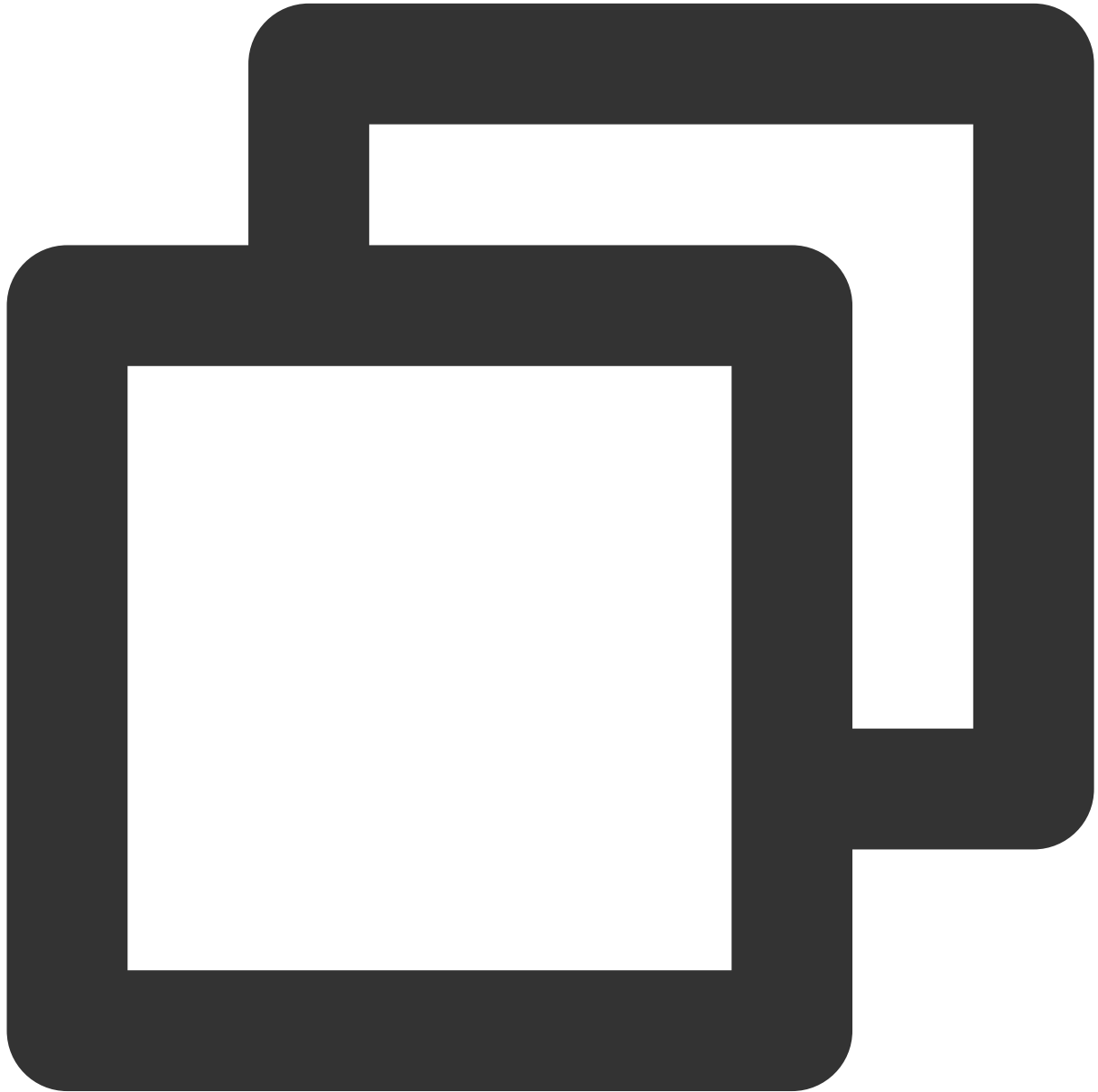
```
<key>Privacy - Camera Usage Description</key>
  <string>The SDK needs to access your camera</string>
```

```
<key>Privacy - Photo Library Usage Description</key>
    <string>The SDK needs to access your album</string>
```

3. Starting the identity verification process

1. Initialize

This API is called when you initialize your app. It is mainly used for some SDK initialization operations.

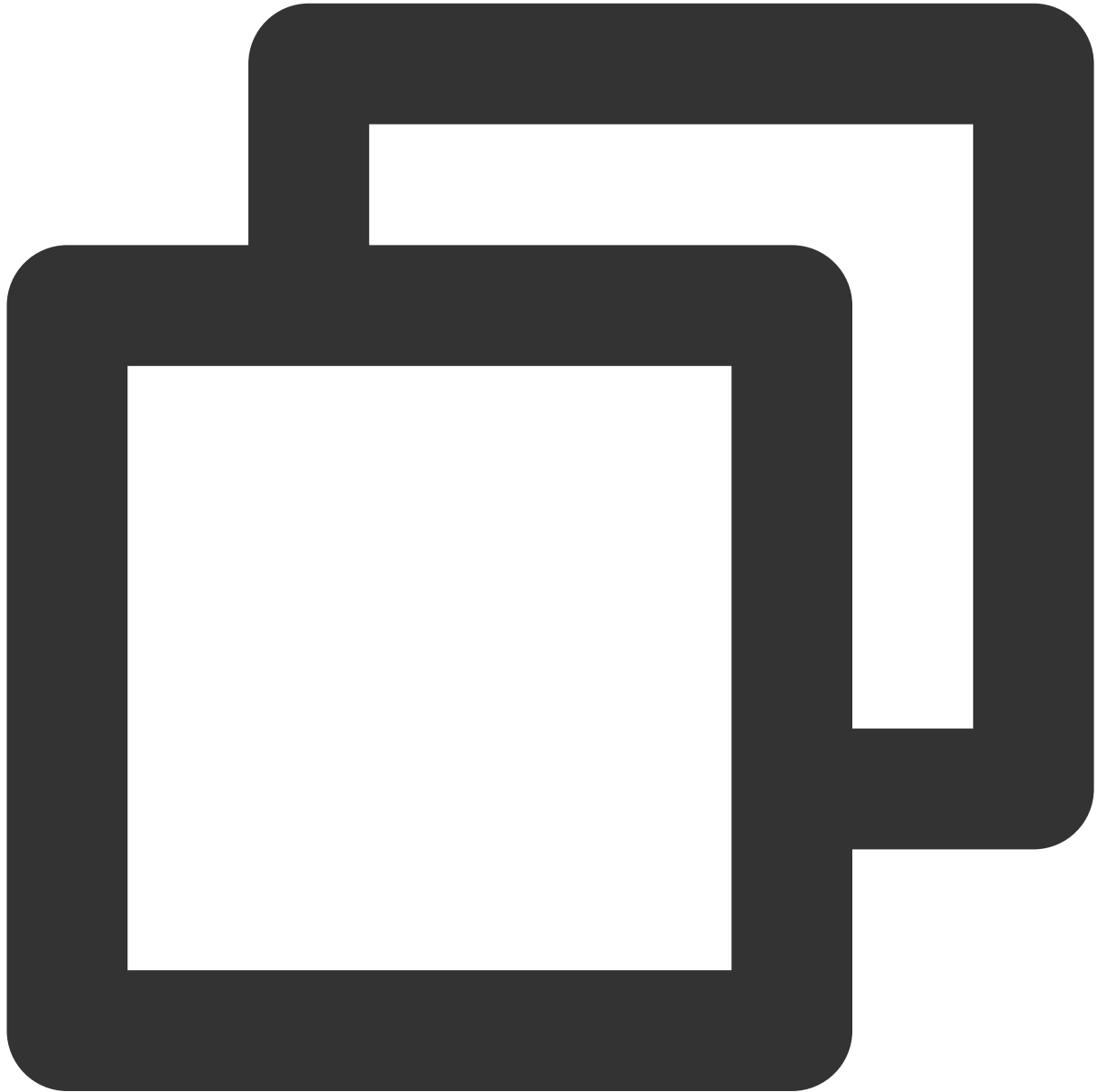


```
#import <HuiYanEKYCVerification/VerificationKit.h>
- (void)viewDidLoad {
    [[VerificationKit sharedInstance] initWithViewController:self];
}
```

```
}
```

2. Start the identity verification process

To start the identity verification process, you just need to call the `startVerifiWithConfig` method with `ekycToken` and some other custom fields set.



```
VerificationConfig *config = [[VerificationConfig alloc] init];  
config.licPath = [[NSBundle mainBundle] pathForResource:@"\" ofType:nil];  
config.languageType = HY_EKYC_EN;  
config.verAutoTimeOut = 30000;// Set the verification timeout period  
config.hyFaceTimeOut = 15000;// Set the timeout period of a single face authenticat
```

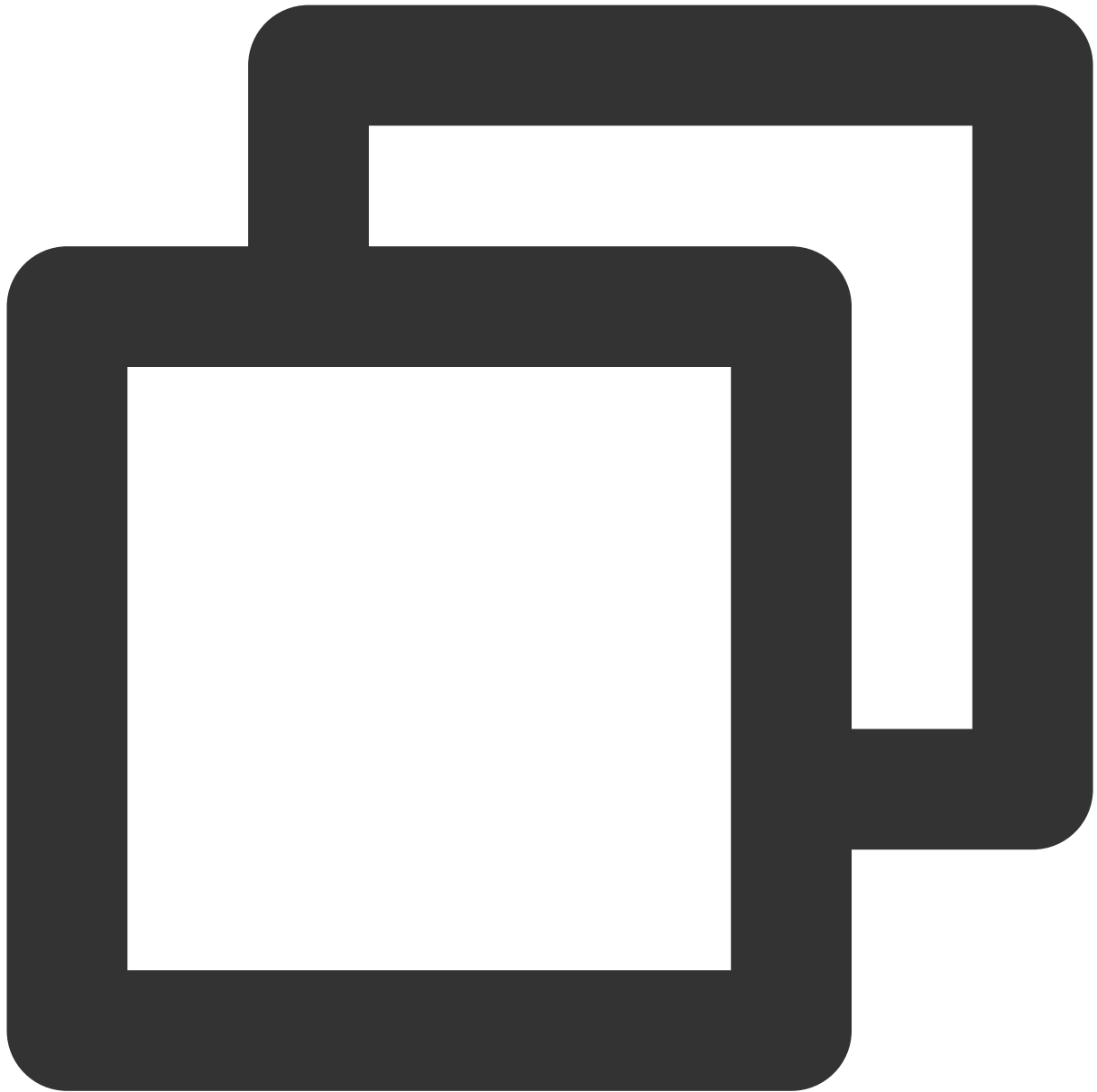
```
config.ekycToken = @"";
[[VerificationKit sharedInstance] startVerifiWithConfig:config withSuccCallback:^(i
    NSLog(@"ErrCode:%d msg:%@",errorCode,resultInfo);
} withFialCallback:^(int errorCode, NSString * _Nonnull errorMsg, id _Nullable res
    NSLog(@"ErrCode:%d msg:%@ extra:%@",errorCode,errorMsg,reserved);
}];
```

ekycToken is the unique credential obtained from the server for this eKYC process.

Note: "eKYC_license.lic" is the license file obtained from the sales rep or customer service, and you need to place it in `Copy Bundle Resources`.

4. Releasing SDK resources

You can call the SDK resource release API after you finish operations and no longer need the SDK.



```
- (void)dealloc {  
    [VerificationKit clearInstance];  
}
```

Note: For the complete code example for iOS, see [iOS demo](#).

SDK API Description

APIs for Android

Last updated : 2023-08-16 15:38:28

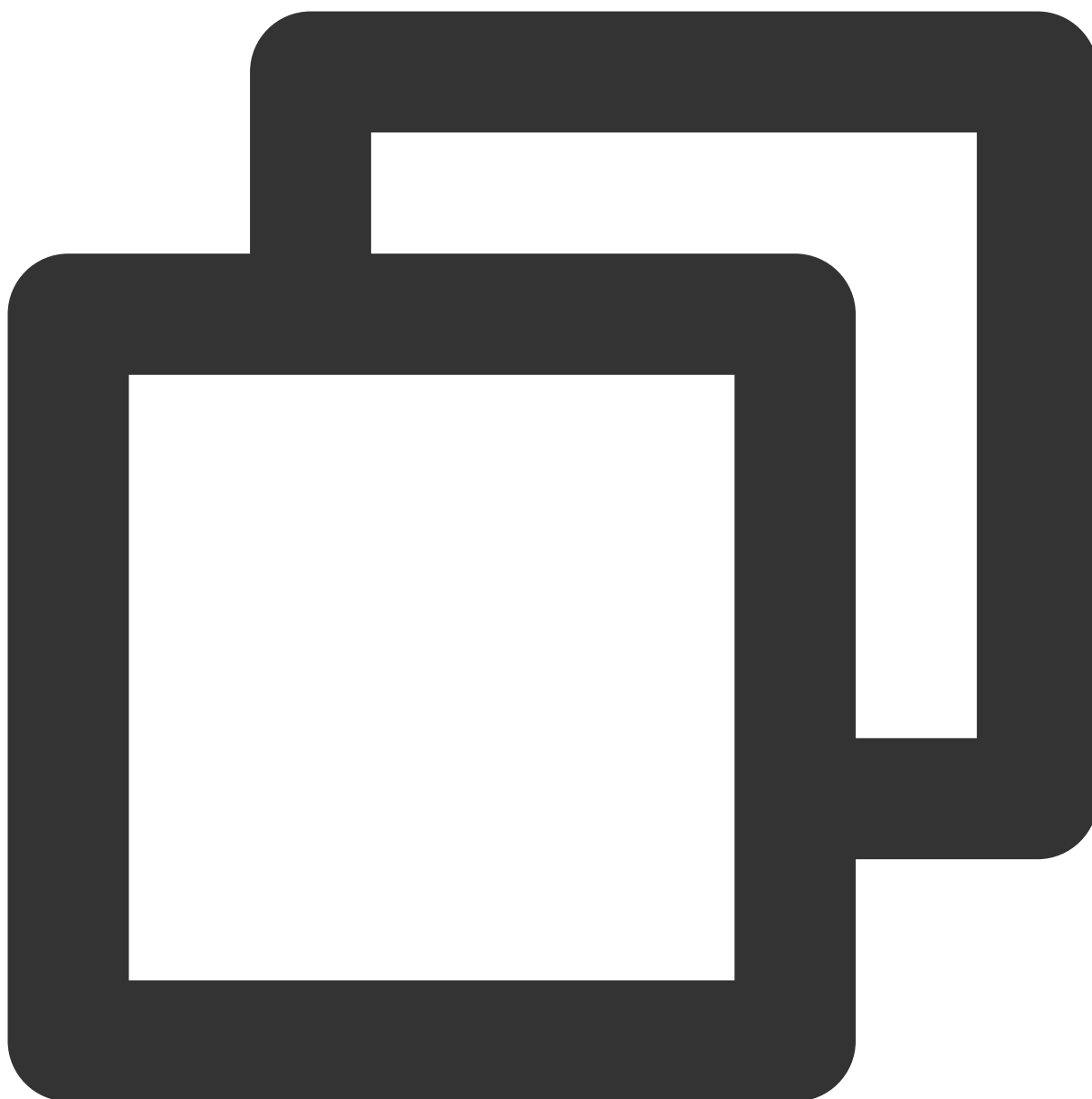
The main API classes used in the Identity Verification (App SDK) for Android are `EkycHySdk` , `EkycHyConfig` , and `EkycHyCallback` . Specific APIs in these classes are described as below.

EkycHySdk

`EkycHySdk` is a class of external APIs for the Identity Verification (App SDK). The main logic is completed with this class.

API	Feature Description
<code>init()</code>	Initializes the SDK.
<code>release()</code>	Releases resources.
<code>startEkycCheck()</code>	Starts the identity verification process.

`init()`



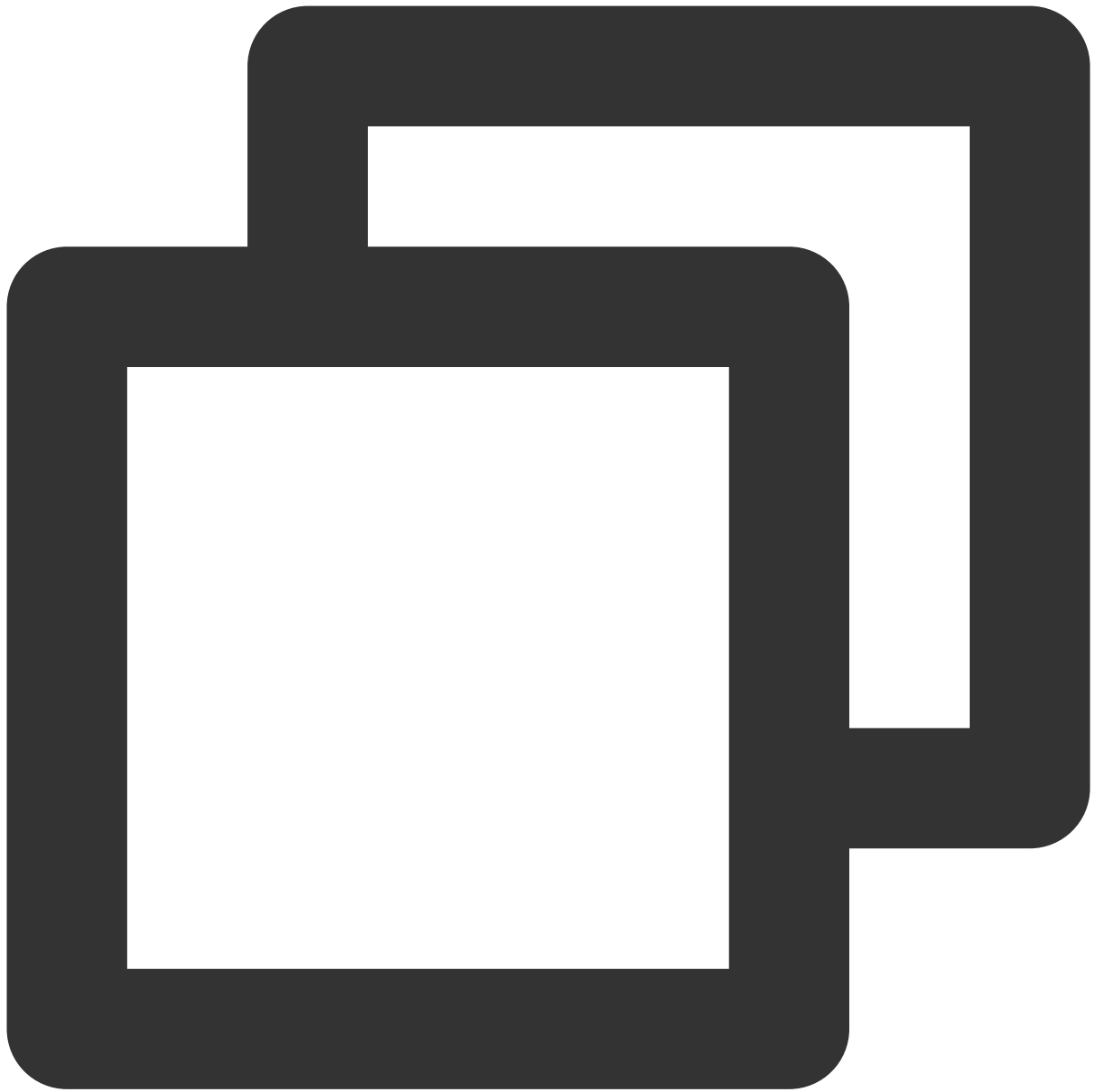
```
public static void init(Context context)
```

Feature description:

This API is used to initialize the Identity Verification (App SDK).

Input parameters:

Type	Parameter	Description
Context	context	App context information

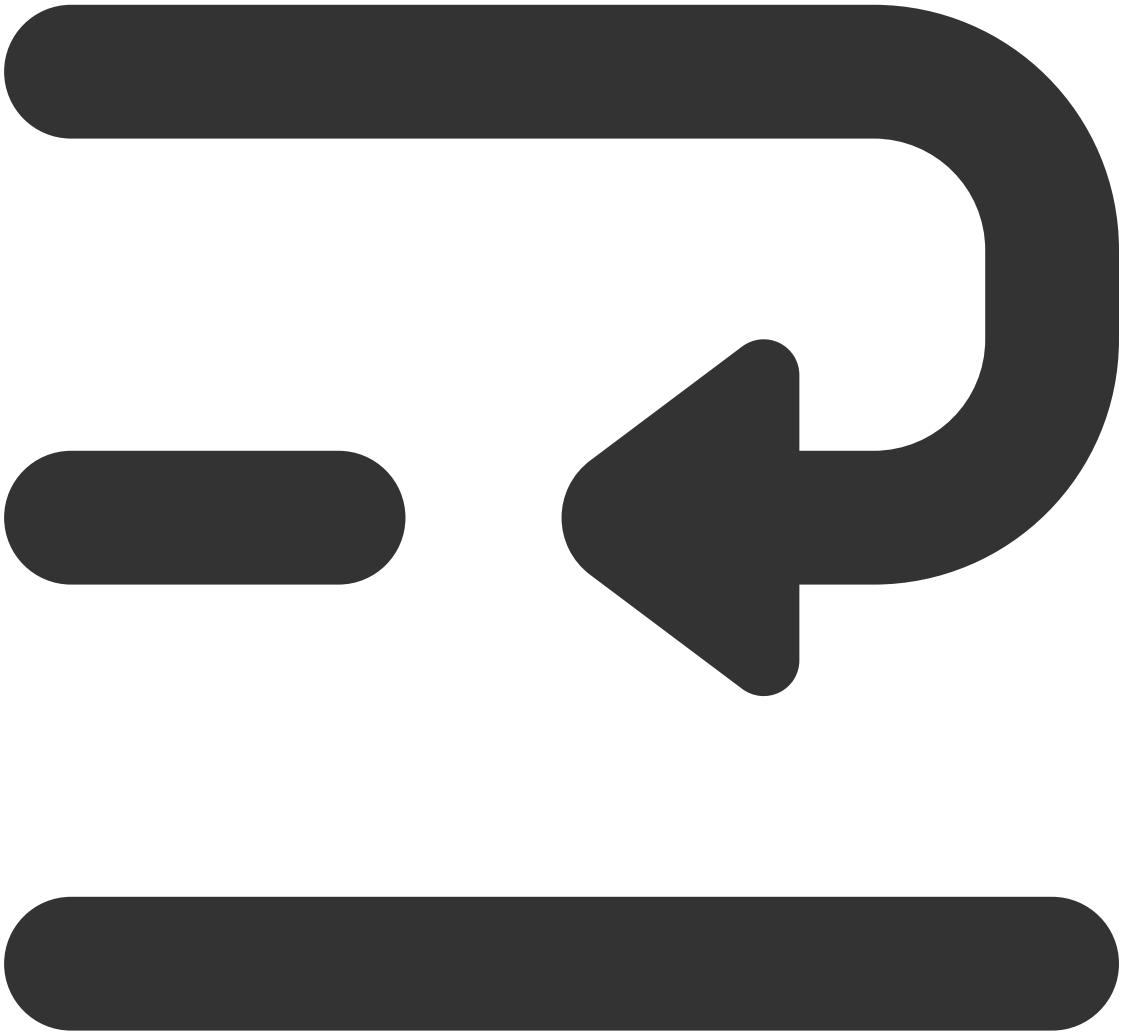
release()

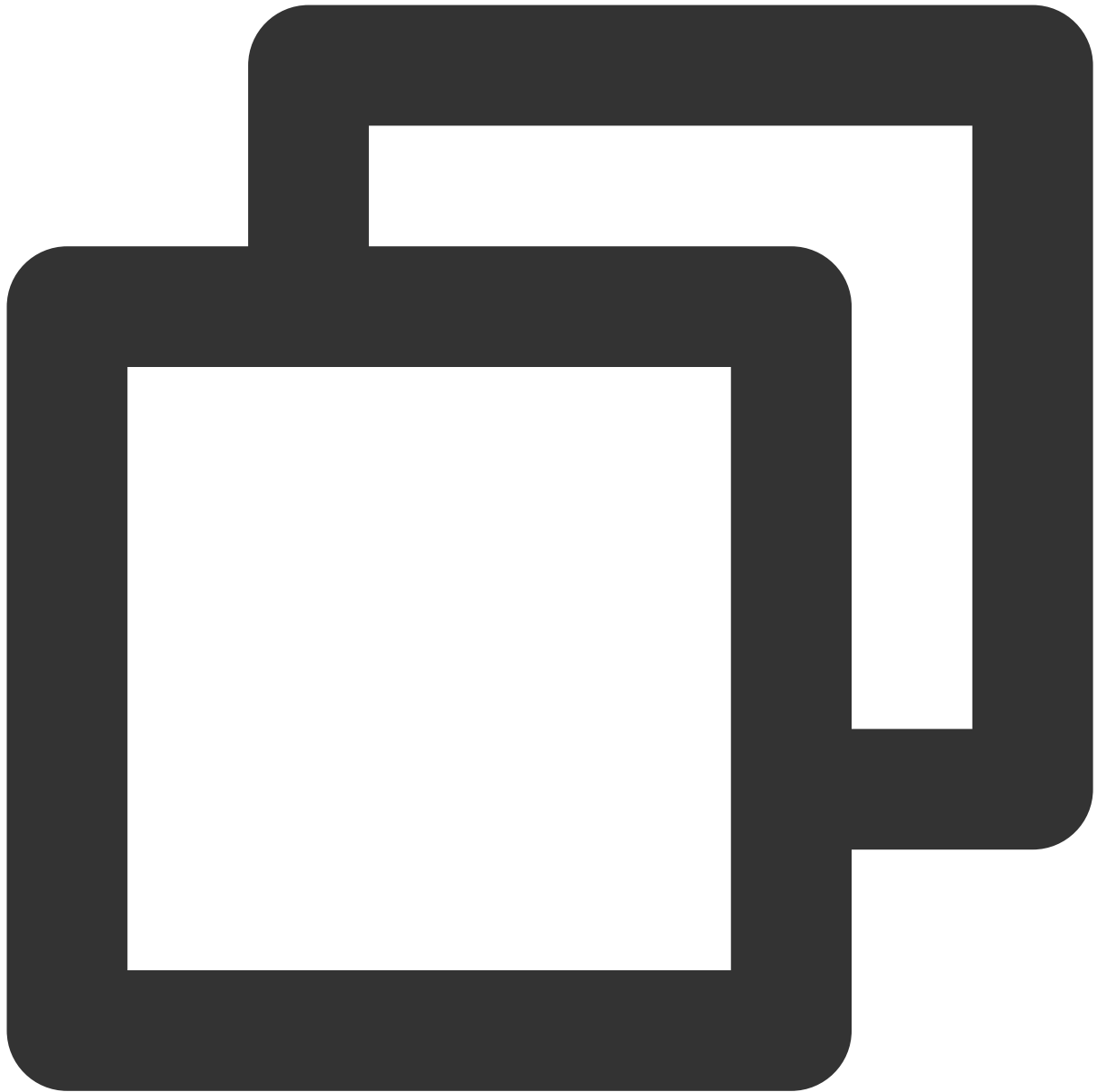
```
public static void release()
```

Feature description:

This API is used to release Identity Verification (App SDK) resources.

startEkycCheck()





```
public static void startEkycCheck(final String ekycToken, EkycHyConfig ekycHyConfig  
    EkycHyCallBack ekycHyCallBack)
```

Feature description:

This API is used to start the identity verification process.

Input parameters:

Type	Parameter	Description
String	ekycToken	Token requested from the server, which is used as the unique business credential for this process

EkycHyConfig	ekycHyConfig	Configuration information for starting this identity verification process
EkycHyCallBack	ekycHyCallBack	API used to receive the verification result callback

EkycHyConfig

`EkycHyConfig` is the configuration entity class used during the Identity Verification (App SDK) startup, which mainly covers the following attributes:

Type	Name	Description	Default Value
String	licenseName	Name of the license file applied for by client for user authorization	Empty
int	verAutoTimeOut	Timeout period for card or certificate verification	20000 milliseconds (20 seconds)
int	ocrAutoTimeout	Timeout period (milliseconds) for automatic capture in OCR_DETECT_AUTO_MANUAL mode. (Set this parameter to at least 5 seconds, and the upper limit is 30 seconds.)	20000 milliseconds (20 seconds)
LanguageStyle	languageStyle	Language setting for this process	LanguageStyle.AUTO
OcrModeType	ocrModeType	OCR mode, which can be manual, auto, or auto+manual	OcrModeType.OCR_DETECT_MA
OcrRegionType	ocrType	Card type	null

OcrRegionType

Type of the document to recognize

Enumerated Value	Description
HK	Hong Kong (China) identity card
ML	Malaysian identity card
PhilippinesDrivingLicense	Philippine driver's license
PhilippinesVoteID	Philippine voters ID card
PhilippinesTinID	Philippine TIN ID card

PhilippinesSSSID	Philippines SSS ID card
PhilippinesUMID	Philippines UMID card
IndonesiaIDCard	Indonesian identity card
MLIDPassport	Passport

LanguageStyle

Language configuration for the default identity verification page

LanguageStyle Value	Description
LanguageStyle.AUTO	Auto
LanguageStyle.ENGLISH	English
LanguageStyle.SIMPLIFIED_CHINESE	Simplified Chinese
LanguageStyle.TRADITIONAL_CHINESE	Traditional Chinese

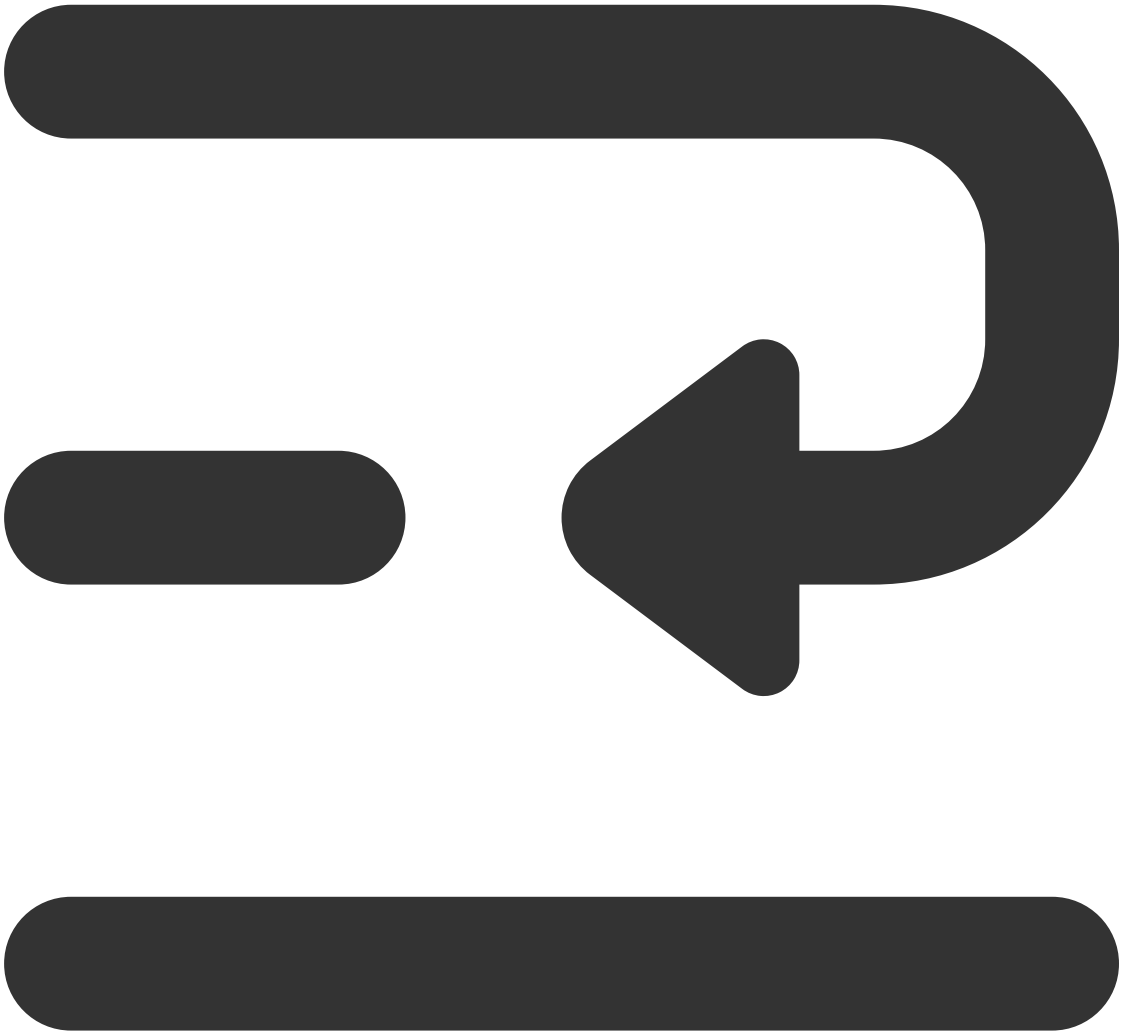
OcrModeType

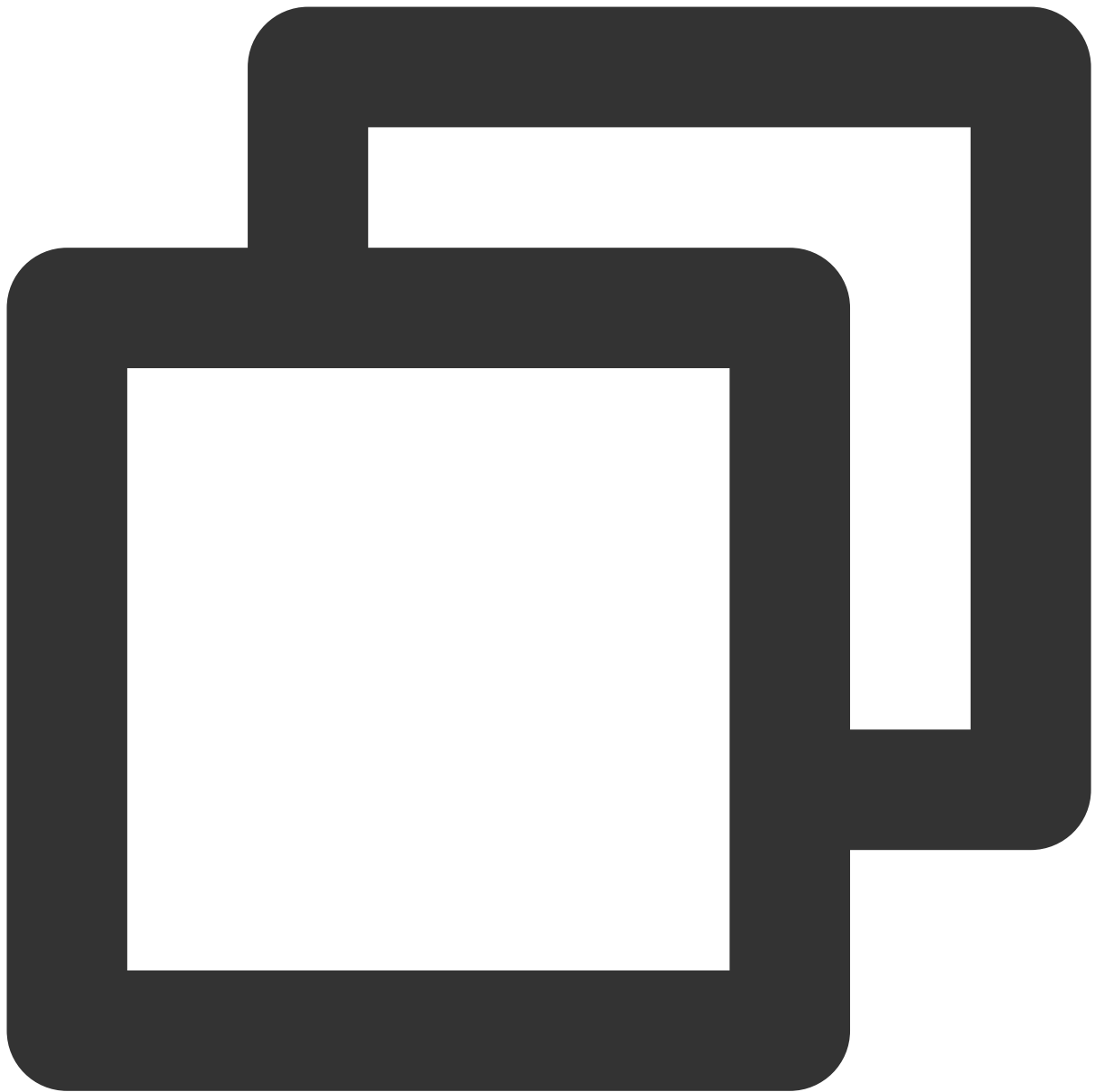
OCR mode

OcrModeType Value	Description
OCR_DETECT_MANUAL	Manual mode
OCR_DETECT_AUTO	Auto mode
OCR_DETECT_AUTO_MANUAL	Auto+Manual mode

EkycHyCallBack

`EkycHyCallBack` is a listener class used to receive the result of the identity verification process.





```
/**
 * Identity verification result callback class
 */
public interface EkycHyCallBack {

    /**
     * Result information on successful recognition
     *
     * @param result: Result data
     */
    void onSuccess(EkycHyResult result);
}
```

```
/**
 * Information on failed identity verification
 *
 * @param errorCode: Error code
 * @param errorMsg: Error message
 * @param ekycToken: The token of this process
 */
void onFail(int errorCode, String errorMsg, String ekycToken);
}
```

Here, **EkycHyResult** is the result object returned after a successful process.

EkycHyResult

EkycHyResult is the result object returned after the identity verification process succeeds.

Type	Name	Description	Default Value
String	ekycToken	The token of this identity verification process, which can be used to pull key data of the process from the server	Empty

Error codes and descriptions

Error Code	Description
12000	The user actively canceled the operation
12001	The network request failed
12002	Error caused by OCR exception
12003	Exception caused by local face detection failure
12004	Invalid token
12005	Failed to perform local card or certificate verification
12006	Failed to initialize the Identity Verification (App SDK)
12007	Failed to start parameter verification
12008	Failed to return the identity verification result
12009	Failed to perform the local identity verification

APIs for iOS

Last updated : 2023-08-10 15:02:45

API description

The main API classes used in the Identity Verification (App SDK) for iOS are `VerificationKit` , `VerificationConfig` , and `VerifiCommDef` . Specific APIs in these classes are described as below.

1.VerificationKit

`VerificationKit` is a class of external APIs for the Identity Verification (App SDK). The main logic is completed with this class.

API	Feature Description
<code>initWithViewController</code>	Initializes the SDK.
<code>clearInstance</code>	Releases resources.
<code>startVerifiWithConfig</code>	Starts the identity verification process.

1.1. initWithViewController



```
// Initialization method  
- (void)initWithViewController:(UIViewController *)viewController;
```

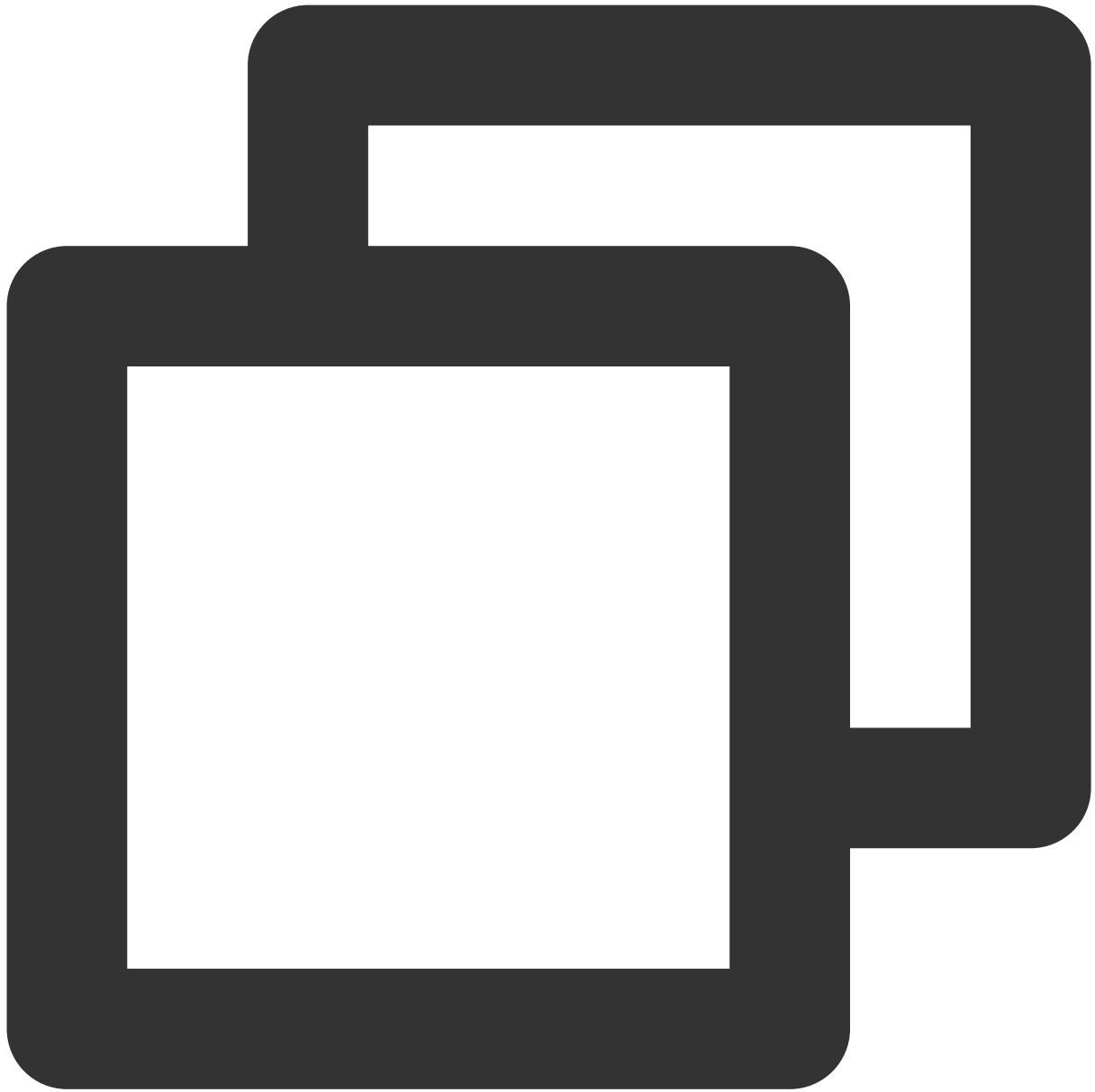
Feature description:

This API is used to initialize the Identity Verification SDK.

Input parameters:

Type	Parameter	Description
UIViewController	viewController	Calls the <code>viewController</code> object of the current SDK page

1.2 clearInstance

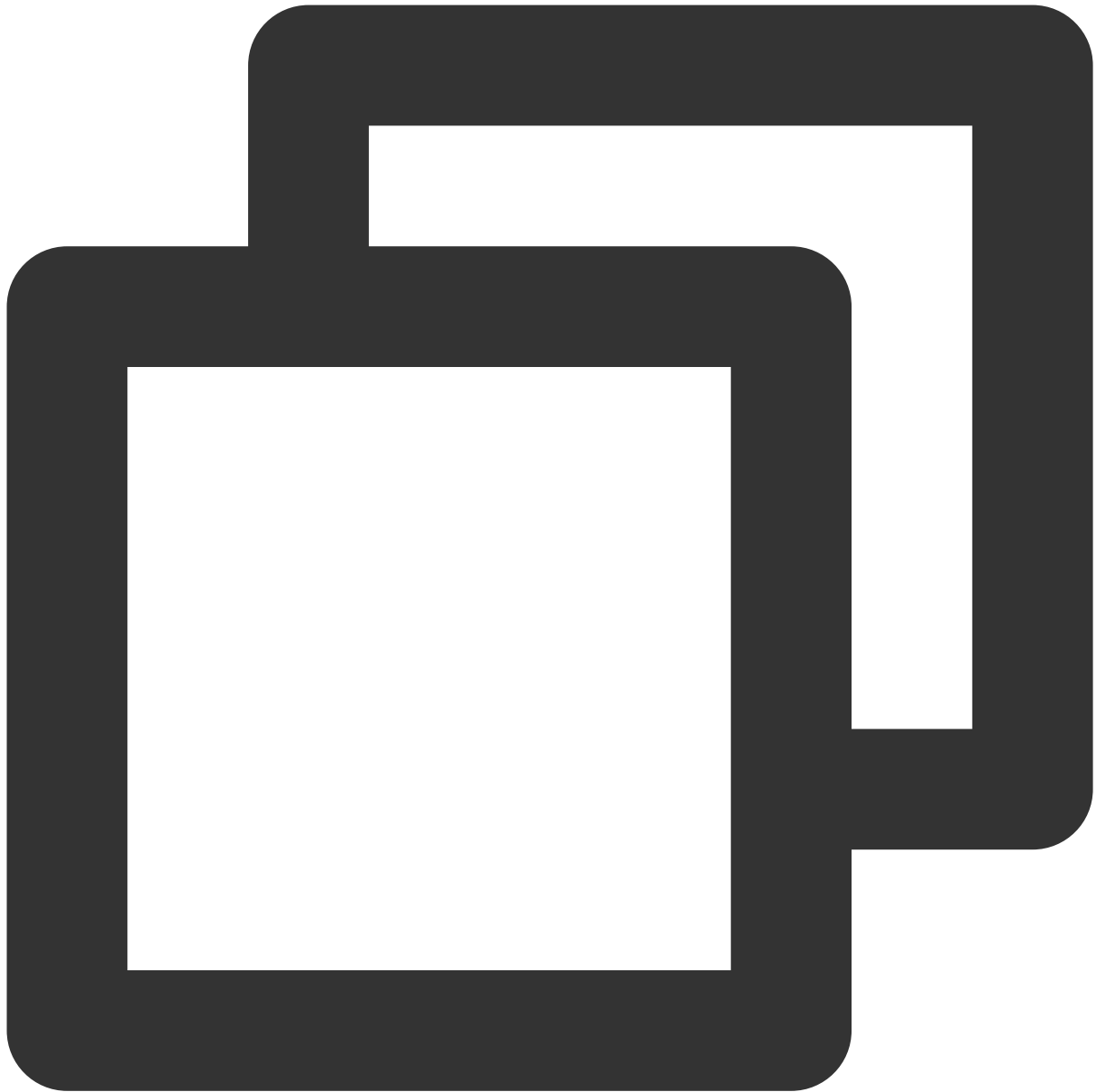


```
/// Clear SDK resources  
+ (void)clearInstance;
```

Feature description:

This API is used to release SDK resources.

1.3 startVerifiWithConfig:



```
/// Start verification
- (void)startVerifiWithConfig:(VerificationConfig *)verifiConfig
    withSuccCallback:(TXYVerifiKitProcessSucceedBlock) succCallback
    withFialCallback:(TXYVerifiKitProcessFailedBlock) failCallback;
```

Feature description:

This API is used to start the identity verification process.

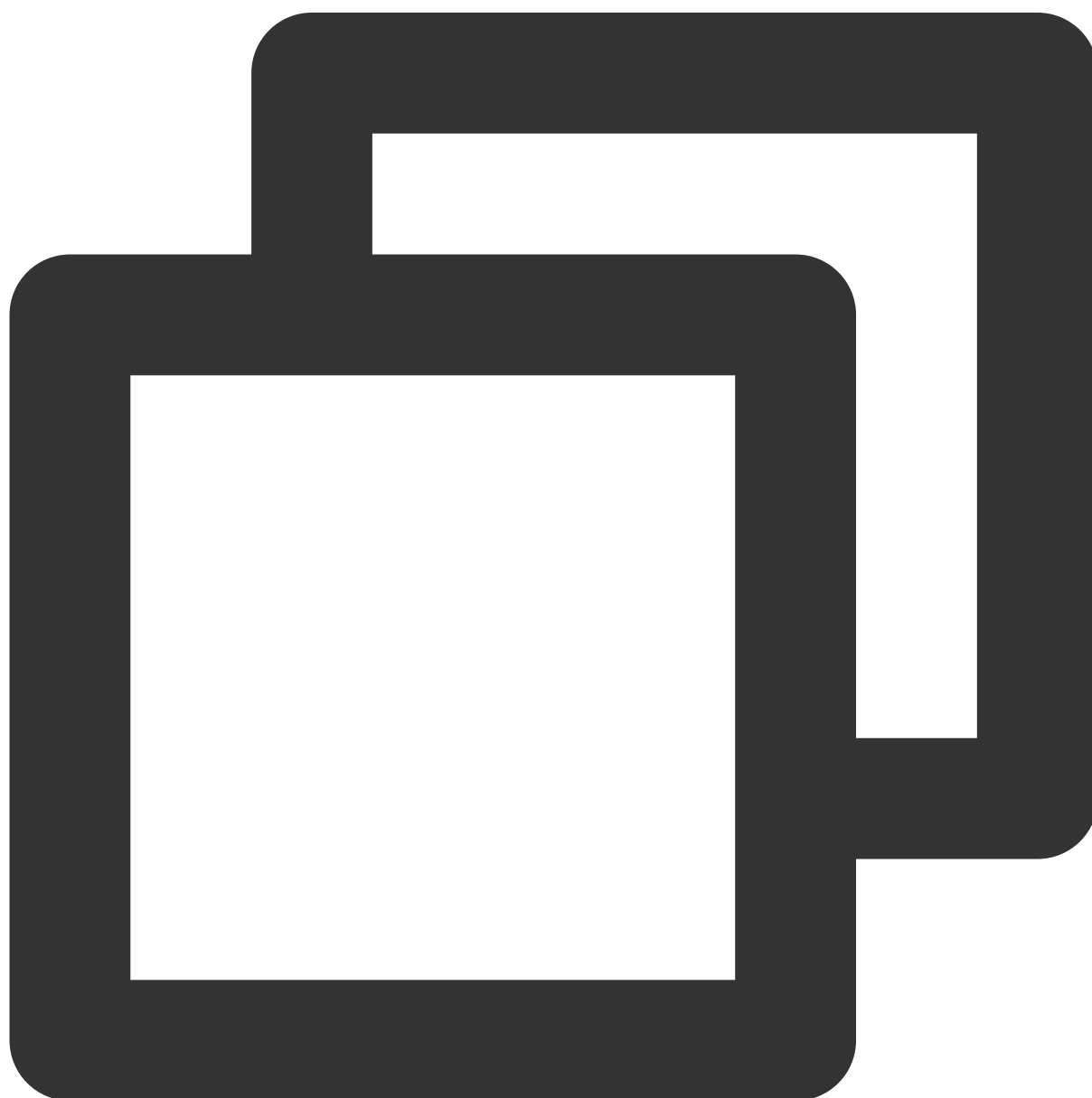
Input parameters:

Type	Parameter	Description

VerificationConfig	verifiConfig	Configuration information for starting this identity verification process
TXYVerifiKitProcessSucceedBlock	succCallback	Callback for successful SDK detection
TXYVerifiKitProcessFailedBlock	failCallback	Callback for failed SDK detection

1.4 TXYVerifiKitProcessSucceedBlock

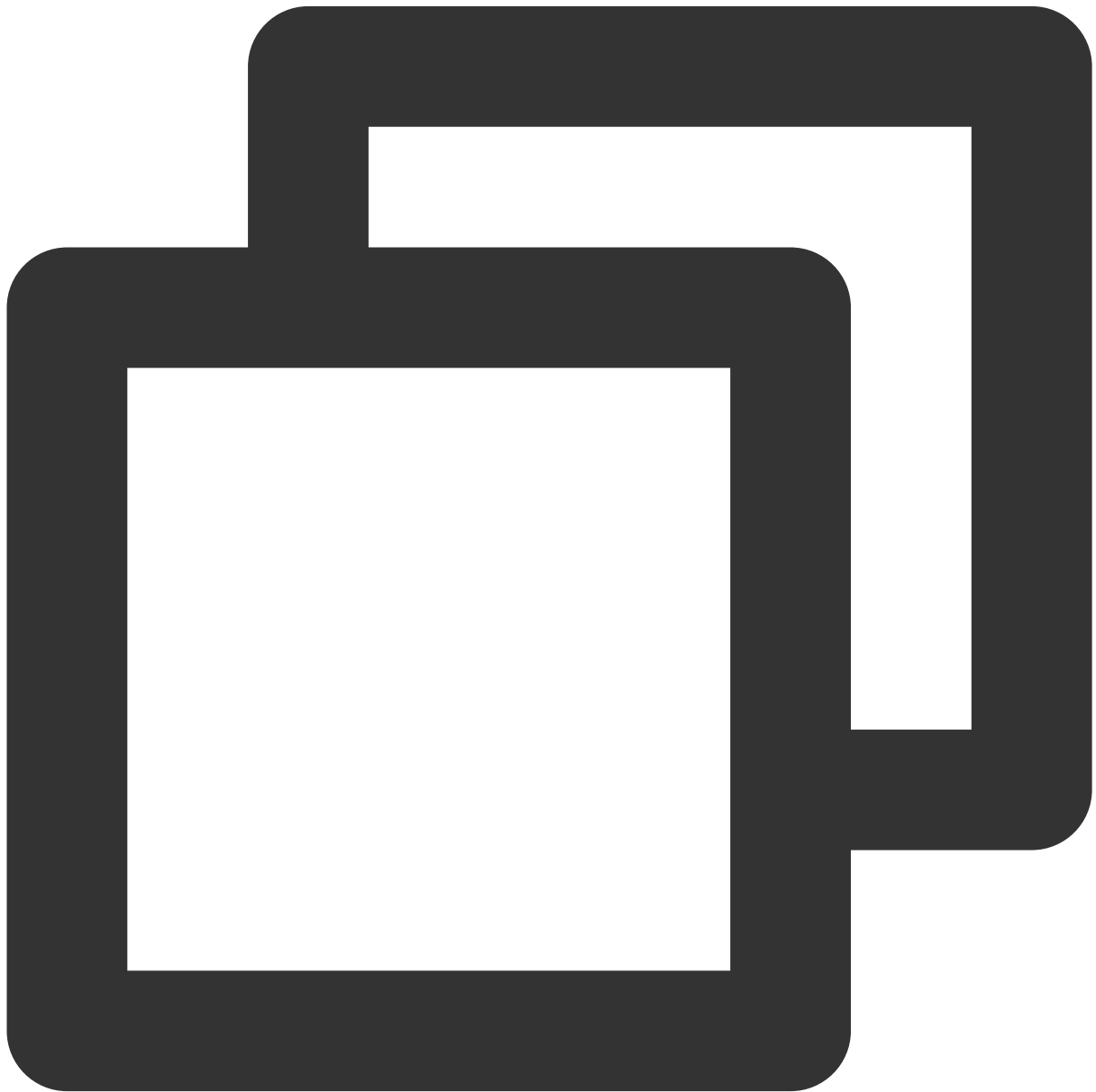
Callback for successful SDK detection




```
/// Callback API for successful detection with the SDKKit  
/// @param errorCode: Error code  
/// @param resultInfo: Information returned by the callback  
/// @param reserved: Reserved  
typedef void (^TXYVerifiKitProcessSucceedBlock)(int errorCode,id _Nonnull resultInf
```

1.5 TXYVerifiKitProcessFailedBlock

Callback for failed SDK detection



```

/// Callback API for detection failure with the SDKKit
/// @param errorCode: Error code
/// @param errorMsg: Error message
/// @param reserved: Reserved
typedef void (^TXYVerifiKitProcessFailedBlock)(int errorCode, NSString *_Nonnull er

```

2.VerificationConfig

VerificationConfig is the configuration entity class used during the SDK startup, which mainly covers the following attributes:

Type	Name	Description	Default Value
NSString	ekycToken	Token requested from the server, which is used as the unique business credential for this identity verification process	Empty
NSString	licPath	Path of the license file applied for by client for user authorization	Empty
long	hyFaceTimeOut	Timeout period for a single action of identity verification	10000 ms (10s)
BOOL	isHiddenAlbum	Whether to hide the OCR album button	No
BOOL	isHiddenFlash	Whether to hide the OCR flashlight button	No
HYEkycLanguageType	languageType	Language setting for this process	DEFAULT (0)
OCRRegionType	ocrRegionType	Type of document to recognize	0
NSString	userUIBundleName	Custom UI bundle name	Nil
NSString	userLanguageBundleName	Custom language bundle name	nil
NSString	userLanguageFileName	Custom language	nil

filename

3.VerifiCommDef

3.1 HYEkycLanguageType

This API provides the language configuration information of the default SDK page.

Type	Description
HY_EKYC_DEFAULT = 0	Auto
HY_EKYC_ZH_HANS	Simplified Chinese
HY_EKYC_ZH_HANT	Traditional Chinese
HY_EKYC_EN	English
HY_EKYC_CUSTOMIZE_LANGUAGE	Custom language. The set custom language bundle (<code>userLanguageBundleName</code>) is used.

3.2 OCRRegionType

Type of document to recognize

Enumerated Value	Description
OCR_TYPE_DEFAULT = 0	Left empty by default
OCR_TYPE_HK	Hong Kong (China) identity card
OCR_TYPE_ML	Malaysian identity card
OCR_TYPE_PV_ID	Philippine driver's license
OCR_TYPE_PDL	Philippine voters ID card
OCR_TYPE_INDONESIA	Indonesian identity card
OCR_TYPE_SINGAPORE	Singapore identity card
OCR_TYPE_PH_TINID	Philippine TIN ID card
OCR_TYPE_PH_SSSID	Philippines SSS ID card
OCR_TYPE_PH_UMID	Philippines UMID card
OCR_TYPE_MLID_PASSPORT	Passport

4. Error codes and descriptions

Error	Error Code	Error Description
HY_SUCCESS	0	Success
HY_VERIFI_FAIL	-1	Detection failure
HY_VERIFI_OCR_FAIL	-2	Document recognition failed
HY_SDK_INNER_ERR	-4	Internal error
HY_INITIALIZATION_PARAMETER_EXCEPTION	310	Parameter initialization exception
HY_BUNDLE_CONFIGURATION_EXCEPTION	311	Bundle configuration exception
HY_YTSDK_CONFIGURATION_EXCEPTION	312	YouTu configuration exception
HY_PLEASE_CALL_FIRST_INIT_API	313	Call the initialization API first
HY_SDK_AUTH_FAILED	314	SDK authorization failure
HY_USER_VOLUNTARILY_CANCELED	315	It is manually canceled by user
HY_YTSDK_LOCAL_AUTH_FAILED	316	Local face detection failure of SDK
HY_CAMERA_OPEN_FAIL	317	Failed to enable the camera
HY_DONOT_SWITCH_APPS	318	Do not switch the application during identity verification
HY_CAMEREA_PERMISSION_EXCEPTION	319	Camera permission exception
HY_SDK_VEDIO_CUT_EXCEPTION	320	Failed to clip the video
HY_LIGHT_DATA_FORMAT_EXCEPTION	321	Invalid reflection data format
HY_GET_REMOTE_DATA_EXCEPTION	322	Error in getting remote data

FAQs

Last updated : 2023-06-12 15:32:39

This document describes FAQs related to the Identity Verification (App SDK) and provides solutions.

Client

Android

1. What should I do if I receive the **Invoke-customs are only supported starting with Android O (--min-api 26)** error after integrating eKYC? Add the following configuration to the `build.gradle` file:



```
// Java 1.8 is supported
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

2. If the integrator uses the obfuscation tool AndResGuard, you can add the following obfuscation configuration:



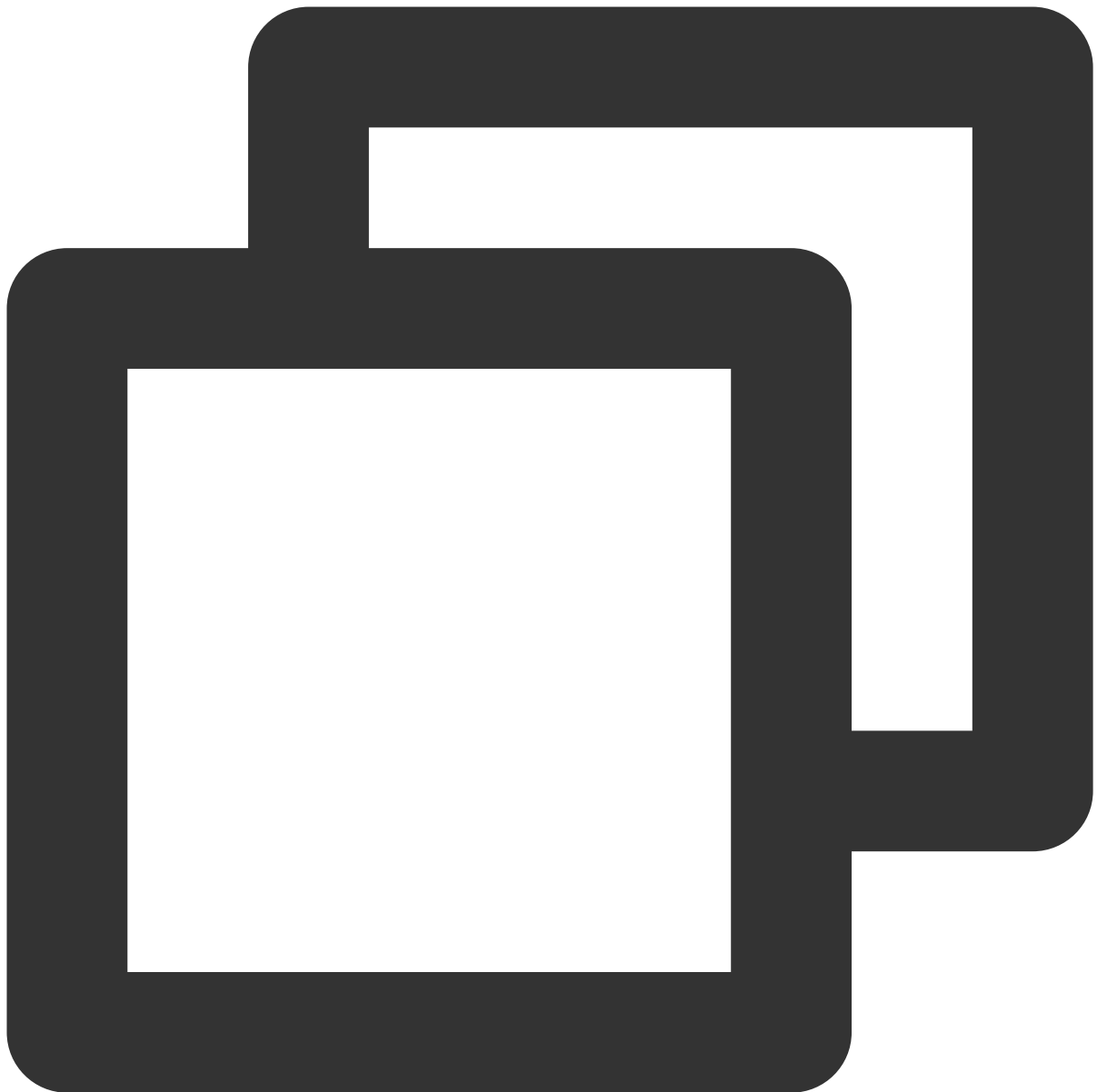
```
// for HuiYanSDK
"R.string.ocr_",
"R.string.rst_",
"R.string.net_",
"R.string.msg_",
"R.string.fl_",
```

3. If Android X reports **android.content.res.Resources\$NotFoundException:from xml type xml resource ID #0x7f0800c3** on devices with an earlier version of system, you can add dependent vector diagrams.



```
// Vector diagrams for earlier versions  
implementation  
'androidx.vectordrawable:vectordrawable:1.1.0'
```

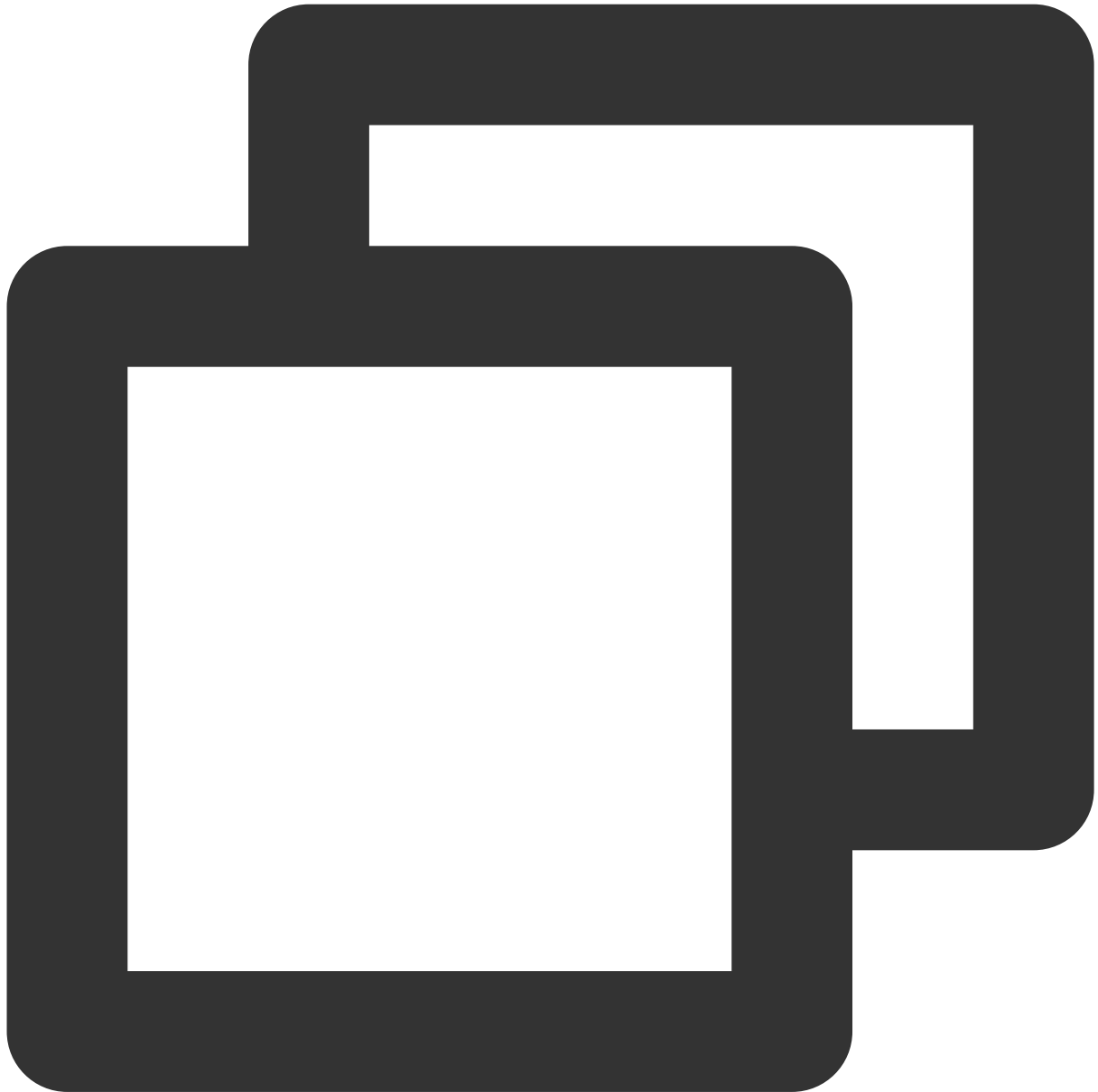
4. If **Android Support** reports the following errors on devices with an earlier version (v6.0 or earlier) of system:



```
android.content.res.Resources$NotFoundException: File
res/drawable/$txy_face_id_logo__0.xml from color state list resource ID #0x7f070001
    at android.content.res.Resources.loadColorStateListForCookie (Resources.java:2749)
    at android.content.res.Resources.loadColorStateList (Resources.java:2749)
    at android.content.res.TypedArray.getColor (TypedArray.java:441)
    at android.content.res.XResources$XTypedArray.getColor (XResources.java:128)
    at android.support.v4.content.res.TypedArrayUtils.getNamedColor (TypedArrayUtils.java:128)
    at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.update (VectorDrawableCompat.java:441)
    at android.support.graphics.drawable.VectorDrawableCompat$VFullPath.inflate (VectorDrawableCompat.java:441)
    at android.support.graphics.drawable.VectorDrawableCompat.inflate (VectorDrawableCompat.java:441)
```

```
at android.support.graphics.drawable.VectorDrawableCompat.createFromXmlInn
at android.support.v7.widget.AppCompatDrawableManager$VdcInflateDelegate.c
```

You need to update the support dependencies to the latest version (v28.0.0):



```
implementation 'com.android.support:appcompat-v7:28.0.0'
// A component library compatible with vector diagrams for earlier versions
implementation 'com.android.support:support-vector-drawable:28.0.0'
implementation 'com.android.support:animated-vector-drawable:28.0.0'
```

iOS

1. What should I do if the SDK crashes and the following log is printed when I enter the SDK: "**reason: '**' - [__NSDictionaryM setObject:forKey:]: key cannot be nil**"? Solution: Add **-ObjC** in "Build Settings" > "Other Linker Flags".
2. What should I do if the following information is displayed during compilation:
Undefined symbol: _vImageConvert_Planar16FtoPlanarF
Undefined symbol: _vImageConvert_PlanarFtoPlanar16F
Solution: Add the system library "Accelerate.framework".
3. What should I do if "**face-tracker-v001 bundle path is nil**" or **HuiYanSDKUI bundle path is nil** is reported?
Solution: Add the two prompted resource files to "Copy Bundle Resources".

Integrating Identity Verification (Mobile HTML5)

Integration Process

Last updated : 2024-01-25 10:41:01

This document introduces the overall integration process of eKYC liveness detection and face comparison (mobile HTML5).

Preparations

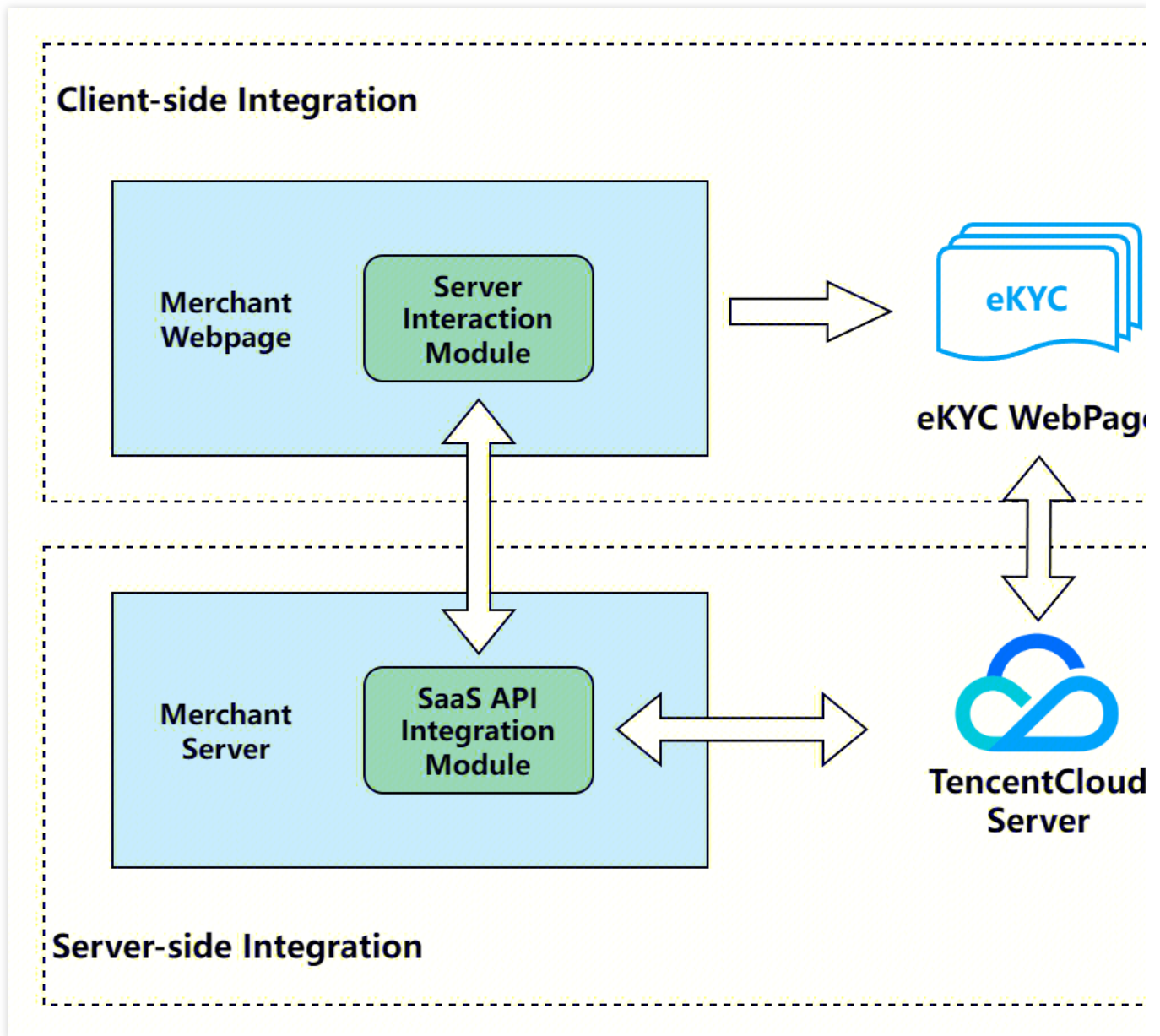
Sign up for a Tencent Cloud account. For more information, see [Signing Up](#).

Complete enterprise identity verification. For more information, see [Enterprise Identity Verification Guide](#).

Log in to the [eKYC console](#).

Overall Architecture

The following figure shows the architecture of eKYC liveness detection and face comparison (mobile HTML5).



Overall Interaction Flow

The following diagram shows the overall interaction logic between customers and Tencent Cloud eKYC. Role descriptions are as follows:

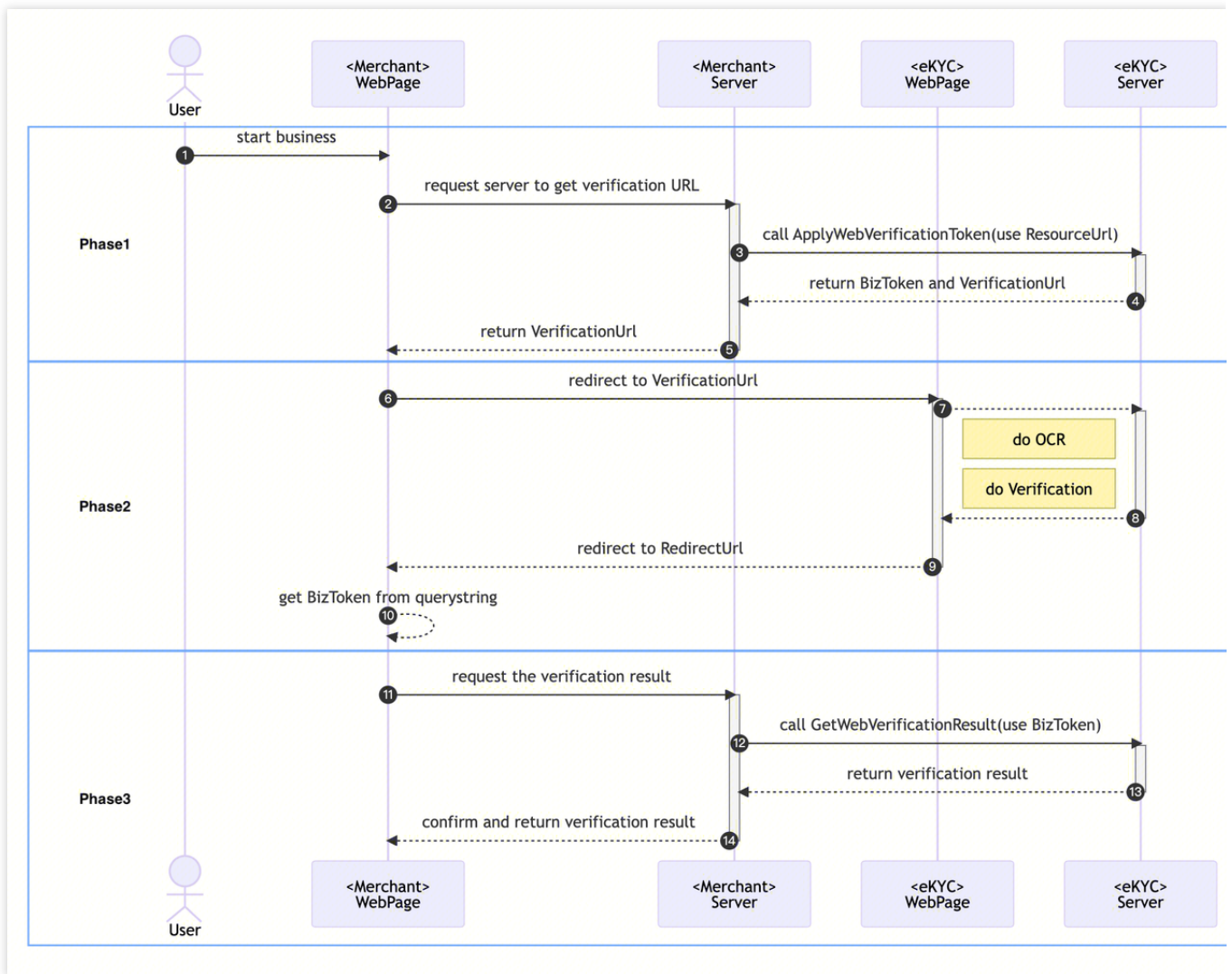
User: H5 users on mobile

Merchant WebPage: customer's frontend page

Merchant Server: customer's backend server

eKYC WebPage: the eKYC frontend page

eKYC Server: the eKYC backend server



The specific recommended interaction flow is as follows:

Phase 1

1. The client service triggers a verification process.
2. Merchant WebPage sends a request to Merchant Server for initiating a verification process.
3. Merchant Server passes in relevant parameters and calls the [ApplyWebVerificationBizTokenIntl](#). Refer to Step 1 of Server Integration.
4. After receiving the request, eKYC Server returns BizToken and VerificationURL of this verification process to Merchant Server.
5. Merchant Server keeps the BizToken and sends VerificationURL to Merchant WebPage.

Phase 2

1. Merchant WebPage redirects to VerificationURL to open eKYC WebPage. Refer to Step 1 of Frontend Integration.
2. The user performs the license OCR and verification process on eKYC WebPage.

3. After verification completed, eKYC Server sends the result to eKYC WebPage, and Merchant WebPage displays the result.
4. After the user clicks Next, eKYC WebPage will redirect to RedirectURL and splice the token parameter on the URL.
5. Merchant WebPage obtains token parameter of the current verification process through the address. Refer to Step 2 of Frontend Integration.

Phase 3

1. Merchant WebPage sends a request to Merchant Server for obtaining verification result information.
2. Merchant Server passes in relevant parameters and calls the [GetWebVerificationResultIntl](#). Refer to Step 2 of Server Integration.
3. After receiving the request, eKYC Server returns details of this verification process to Merchant Server.
4. Merchant Server returns the result to Merchant WebPage which will perform subsequent business processes according to the result.

Server Integration

1. Call the interface [ApplyWebVerificationBizTokenIntl](#) to generate verification URL (corresponding to Stage 1)

Call [ApplyWebVerificationBizTokenIntl](#) to obtain **BizToken** and VerificationURL, corresponding to No.3 of the flow diagram.

RedirectURL: the Web address redirected after verification, including protocol header, hostname, and path, e.g., `https://www.tencentcloud.com/products/faceid`. After the verification process completed, the redirect address will be spliced to BizToken of the current process in the format as follows:

```
https://www.tencentcloud.com/products/faceid? token={BizToken} .
```

Extra: the service pass-through parameter, up to 1000 characters. It will be returned in the [GetWebVerificationResultIntl](#) interface. It can be omitted if it is not necessary.

Config: the relevant configuration of custom verification page.

AutoSkip: whether to skip the result page and automatically jump to RedirectURL when verification succeeds. The default is false.

CheckMode: the detection mode. This parameter is required. Parameter values are as follows: 1: OCR+ liveness detection + face comparison.

IDCardType: the license type that supports identification. This parameter is required. The license supported currently are as follows:

HKIDCard: Hong Kong (China) ID Card

MLIDCard: Malaysian ID Card

IndonesialIDCard: Indonesian ID Card

PhilippinesVoteID: Philippine Voter Card

PhilippinesDrivingLicense: Philippines Driving License

PhilippinesTinID: Philippines TinID

PhilippinesSSSID: Philippines SSSID

PhilippinesUMID: Philippines UMID

InternationalID Passport: Passport of Hong Kong, Macao, Taiwan (China), and other countries

IndonesiaDrivingLicense: Indonesian Driving License

ThailandIDCard: Thailand ID Card

ThailandDrivingLicense: Thailand Driving License

MLDrivingLicense: Malaysia Driving License

SingaporeIDCard: Singapore ID Card

SingaporeDrivingLicense: Singapore Driving License

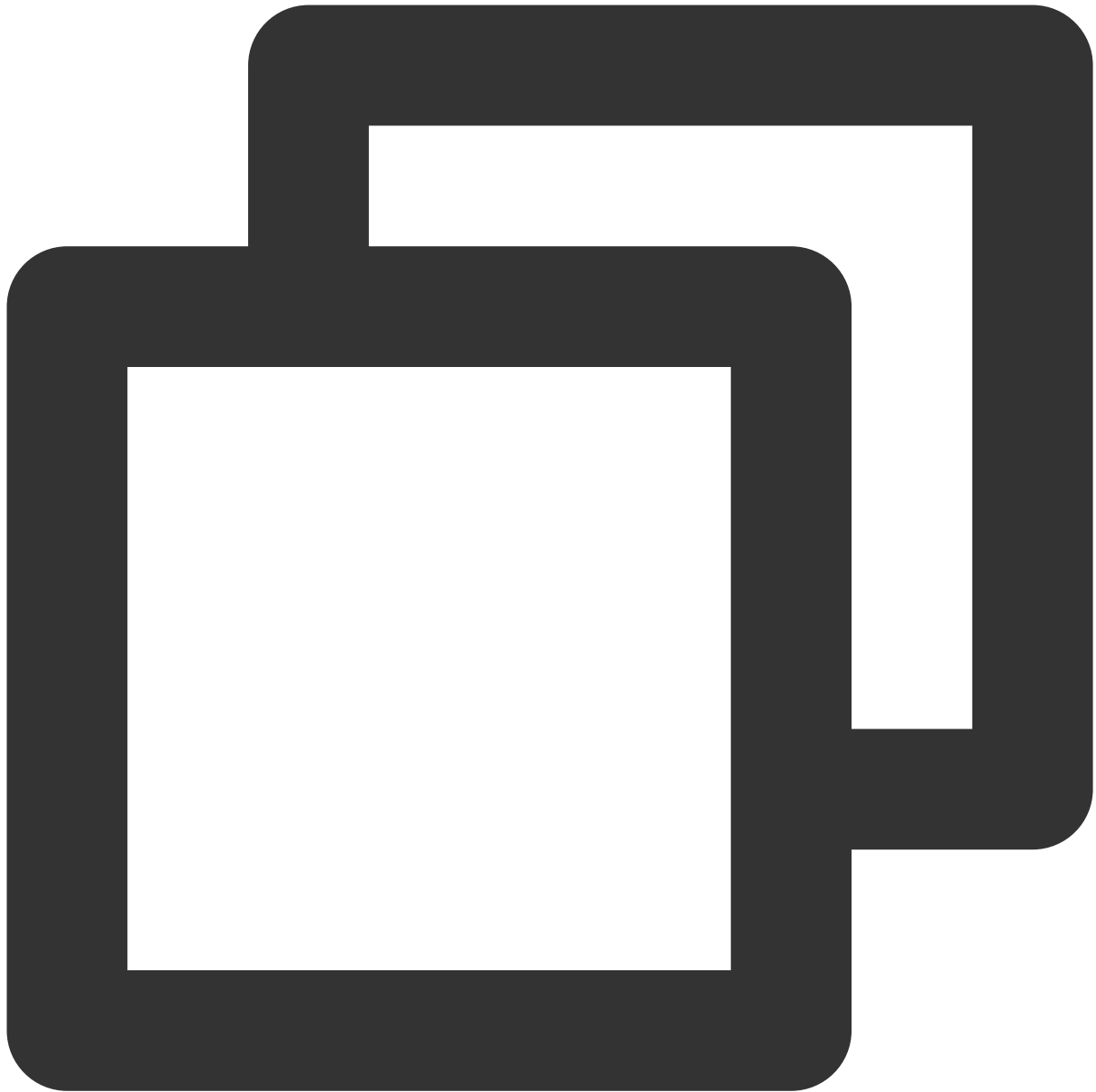
Japanese ID Card: Japanese ID Card

Japanese Driving License

PhilippinesIDCard: Universal Philippine ID Card

DisableCheckOcrWarnings: whether to disable license alarms. The default is false (alarm detection is enabled). When enabled, the identity authentication process will be intercepted according to the license alarm. If you need to use license authentication function, please [Contact Us](#).

Code example of calling interfaces:



```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.faceid.v20180301.FaceidClient;
import com.tencentcloudapi.faceid.v20180301.models.*;;

public class ApplyWebVerificationBizTokenIntl
{
    public static void main(String [] args) {
        try{
```

```

// Instantiate an authentication object. The secretId and secretKey of
// Obtain the key at https://console.tencentcloud.com/cam/capi
Credential cred = new Credential(
    "TENCENTCLOUD_SECRET_ID",
    "TENCENTCLOUD_SECRET_KEY"
);
// Instantiate an http option, optional. Skip this if there are no spec
HttpProfile httpProfile = new HttpProfile();
httpProfile.setEndpoint("faceid.ap-guangzhou.tencentcloudapi.woa.com");
// Instantiate a client option, optional. Skip this if there are no spe
ClientProfile clientProfile = new ClientProfile();
clientProfile.setHttpProfile(httpProfile);
// Instantiate the client object of the product to be requested. The cl
FaceidClient client = new FaceidClient(cred, "ap-singapore", clientProf
// Instantiate a request object, one for each interface.
ApplyWebVerificationBizTokenIntlRequest req = new ApplyWebVerificationB
req.setRedirectURL("https://www.tencentcloud.com/products/faceid");
WebVerificationConfigIntl webVerificationConfigIntl = new WebVerificati
webVerificationConfigIntl.setCheckMode(1L);
webVerificationConfigIntl.setIDCardType("HKIDCard");
req.setConfig(webVerificationConfigIntl);
// The "resp" returned is an instance of ApplyWebVerificationBizTokenIn
ApplyWebVerificationBizTokenIntlResponse resp = client.ApplyWebVerifica
// The string response package output in json format.
System.out.println(ApplyWebVerificationBizTokenIntlResponse.toJsonStrin
String bizToken = resp.getBizToken();
String verificationURL = resp.getVerificationURL();
System.out.printf("BizToken: %s, VerificationURL: %s", bizToken, verifi
} catch (TencentCloudSDKException e) {
    System.out.println(e.toString());
}
}
}
}

```

2. Confirm results of the current verification process (corresponding to Stage 3)

After the verification completed, Merchant Frontend notifies Merchant Server to obtain the result of this verification. Merchant Server calls the [GetWebVerificationResultIntl](#) to return the result to frontend page, corresponding to No.12 of the flow diagram.

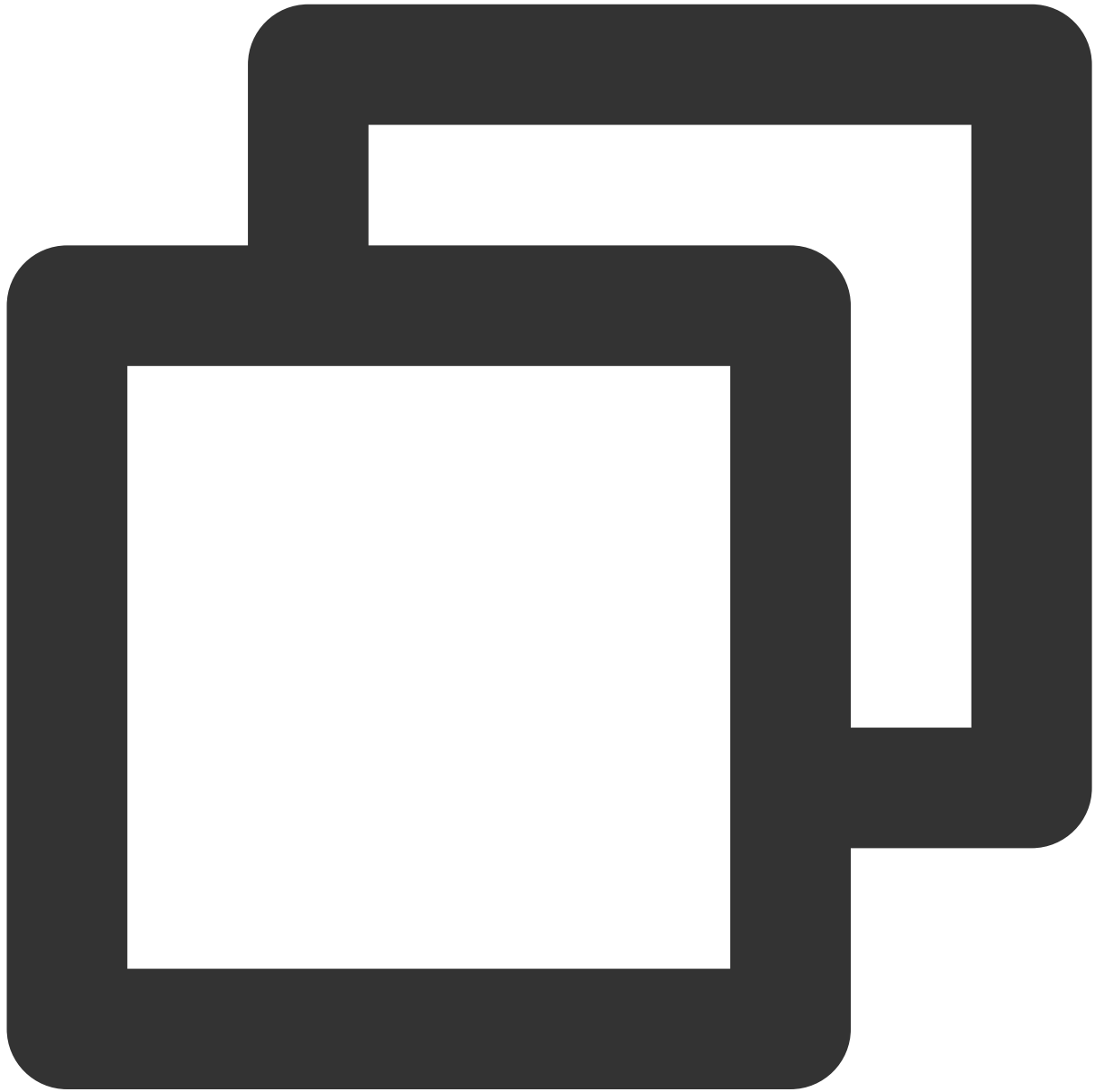
The license OCR result of this verification can be obtained from the responsive "OCRResult" field. The license OCR process can be regarded as success if the "OCRResult" field is not null, and the information of the corresponding certificate can be obtained. In other cases, it can be regarded as failure. For more information about the "OCRResult" field, refer to [OCRResult](#) documentation.

The result of this verification shall be based on the information returned by this interface. When the responsive "ErrorCode" field is 0, this verification process is deemed to have passed. In other cases, it is deemed to have failed.

For error code list, refer to [Liveness Detection and Face Comparison \(Mobile HTML5\) Error Codes](#).

BizToken: the BizToken generated by the ApplyWebVerificationBizTokenIntl, a unique identifier of the current verification process.

Code example of calling interfaces:



```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.faceid.v20180301.FaceidClient;
```

```
import com.tencentcloudapi.faceid.v20180301.models.*;

import java.util.Arrays;

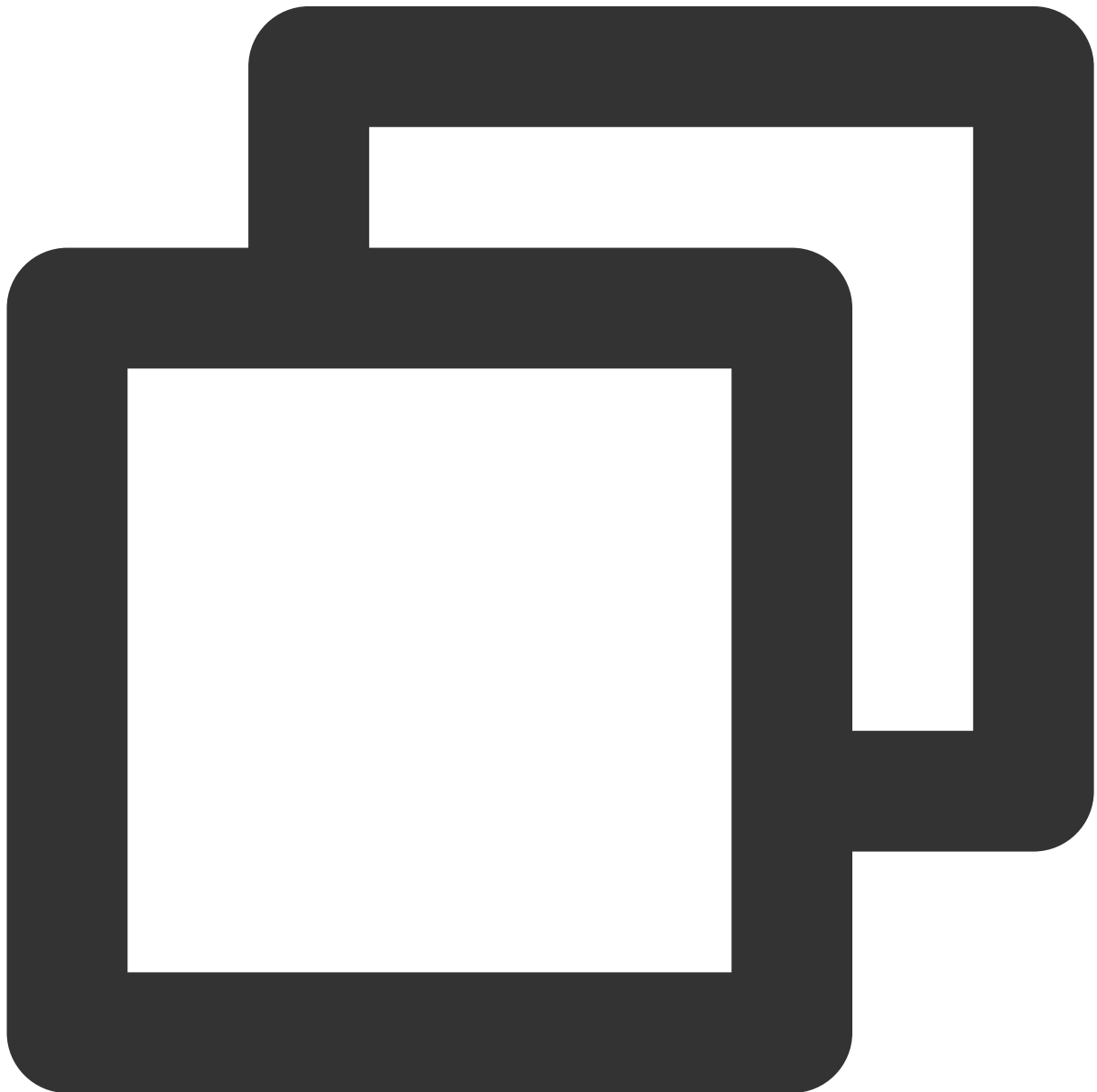
public class GetWebVerificationResultIntl {
    public static void main(String [] args) {
        try{
            // Instantiate an authentication object. The secretId and secretKey of
            // Obtain the key at https://console.tencentcloud.com/cam/capi
            Credential cred = new Credential(
                "TENCENTCLOUD_SECRET_ID",
                "TENCENTCLOUD_SECRET_KEY"
            );
            // Instantiate an http option, optional. Skip this if there are no spec
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("faceid.ap-guangzhou.tencentcloudapi.woa.com");
            // Instantiate a client option, optional. Skip this if there are no spe
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            // Instantiate the client object of the product to be requested. The cl
            FaceidClient client = new FaceidClient(cred, "ap-singapore", clientProf
            // Instantiate a request object, one for each interface.
            GetWebVerificationResultIntlRequest req = new GetWebVerificationResultI
            req.setBizToken("xxx"); // Enter the BizToken returned from ApplyWebVer
            // The "resp" returned is an instance of GetWebVerificationResultIntlRe
            GetWebVerificationResultIntlResponse resp = client.GetWebVerificationRe
            // The string response package output in json format.
            System.out.println(GetWebVerificationResultIntlResponse.toJsonString(re
            Long errorCode = resp.getErrorCode();
            String errorMsg = resp.getErrorMsg();
            // For details of the "OCRResult" field, please refer to OCRResult rela
            OCRResult[] ocrResult = resp.getOCRResult();
            if (errorCode == 0) {
                // Verification passed.
                System.out.println("Success");
                System.out.printf("OCRResult:%s", Arrays.toString(ocrResult));
            }else {
                // Verification failed.
                System.out.printf("Fail: %s\\n", errorMsg);
            }
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

Frontend Integration

1. Obtain VerificationURL and redirect to initiate the verification process (corresponding to Stage 2)

The client frontend page obtains the VerificationURL requested by the server and redirects to enter the verification process. The user completes liveness comparison process according to the prompt, corresponding to No.6 of the flow diagram.

Code example:



```
// Obtain VerificationURL from server.  
const VerificationURL = 'https://sg.faceid.qq.com/reflect/? token=*****';  
// Redirect from frontend page.  
window.location.href = VerificationURL;
```

2. Obtain BizToken from callback address, and request verification result from backend (corresponding to Stage 2)

After verification, the page will jump to RedirectURL which will splice **BizToken** parameters of the current process. The BizToken parameters can be obtained by parsing RedirectURL to pull the result information of this liveness comparison, corresponding to No.12 of the flow diagram.

Code example:



```
// Obtain RedirectURL
const RedirectURL = "https://*****? token={BizToken}";

// Parse to obtain BizToken parameter of RedirectURL, which is used to pull result
const bizToken = getURLParameter(RedirectURL, "token");
if (bizToken) {
    // Use bizToken to pull result information of this liveness comparison.
}

/**
 * Parse url parameters
```

```
/* @params url    To query url
/* @params variable To query parameters
*/
function getUrlParameter(url, variable) {
  const query = url.split('? ')[1] || '';
  const vars = query.split('&');
  for (let i = 0; i < vars.length; i++) {
    const pair = vars[i].split('=');
    if (pair[0] == variable) {
      return pair[1];
    }
  }
  return (false);
}
```


Liveness Detection and Face Comparison (Pure API) Integration Process

Last updated : 2023-11-29 16:00:14

This article describes the overall access process of liveness face comparison in API mode. You can integrate it using any language you are familiar with because this API is language independent. However, before using API to integrate liveness face comparison services, you'd better be able to read and understand [Connecting to Tencent Cloud API](#), which tells you how to use Tencent Cloud SDK to simplify the API access process. In this article, we only demonstrate how to integrate our APIs using Java API SDK.

Preparing for Access

Register a Tencent Cloud enterprise account. Refer to [Signing Up](#).

Complete the enterprise identity verification. Refer to [Enterprise Identity Verification Guide](#).

Log in to FaceID console to [Open Service](#).

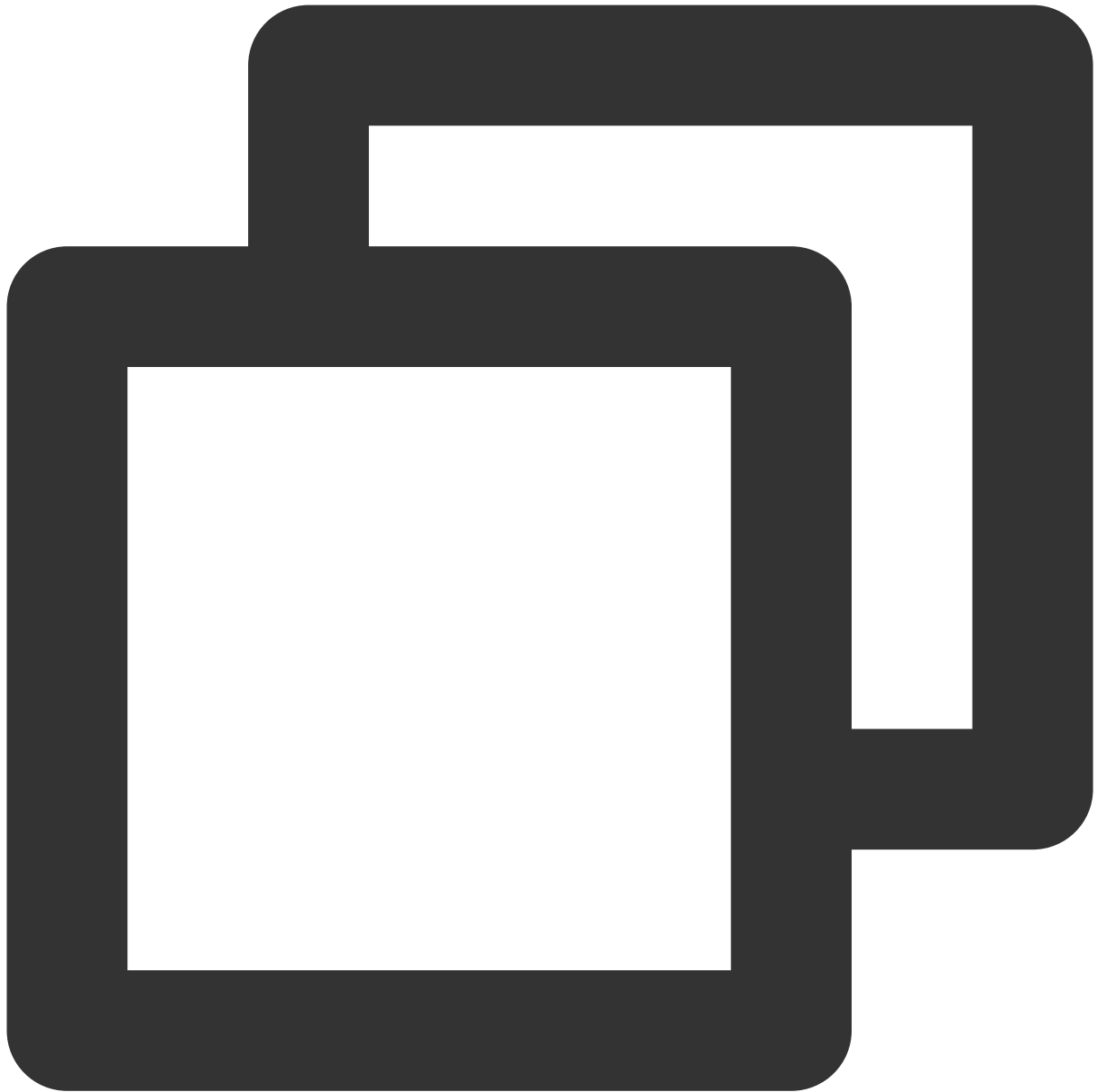
Obtain API access [key](#).

Introducing Tencent Cloud API SDK



```
<dependency>
  <groupId>com.tencentcloudapi</groupId>
  <artifactId>tencentcloud-sdk-java-intl-en</artifactId>
  <version>3.0.798</version>
</dependency>
```

Initializing and Configuring API SDK Client



```
// Instantiate an authentication object. The Tencent Cloud account `secretId` and `
Credential cred = new Credential("secretId", "secretKey");

// Instantiate the client object of the requested product
ClientProfile clientProfile = new ClientProfile();
clientProfile.setSignMethod(ClientProfile.SIGN_TC3_256);

FaceidClient client = new FaceidClient(cred, "ap-singapore", clientProfile);
```

Calling the CompareFaceLiveness API



```
// Step 1: Instantiate the request object and provide necessary parameters
CompareFaceLivenessRequest request = new CompareFaceLivenessRequest();
request.setLivenessType("SILENT");
request.setImageBase64(getBase64(cmd.getOptionValue(IMAGE_PATH)));
request.setVideoBase64(getBase64(cmd.getOptionValue(VIDEO_PATH)));

// Step 2: Call the Tencent Cloud API through FaceIdClient
CompareFaceLivenessResponse response = client.CompareFaceLiveness(request);

// Step 3: Process the Tencent Cloud API response and construct the return object
System.out.println(CompareFaceLivenessResponse.toJsonString(response));
```

Usage Example

You can view full sample source code at GitHub repository:

[CompareFaceLiveness](#)

API Documentation

You can view full API documentation on Tencent Cloud official website:

[CompareFaceLiveness](#)

Other Guide

Quick API Run

Last updated : 2023-06-06 16:21:21

Overview

This document describes how to call Tencent Cloud eKYC APIs through API 3.0 Explorer and integrate SDKs in the corresponding programming language into your project after you purchase the eKYC service. You can access eKYC APIs quickly in the following steps.

Prerequisites

You have entered the [API 3.0 Explorer](#) page.

Directions

The following steps use the calling of the [ApplySdkVerificationToken](#) API as an example.

Step 1.

On the [API 3.0 Explorer](#) page, select `ApplySdkVerificationToken` from the left sidebar, enter required parameters, initiate the call, and get the response.

The screenshot displays the Tencent Cloud API Explorer for the `ApplySdkVerificationToken` API. The interface includes a sidebar with a list of APIs, a main panel for the selected API, and a right-hand panel for online invocation. The `ApplySdkVerificationToken` API is selected, and its version is `2018-03-01`. The `Input Parameters` section is highlighted with a red box, showing the following fields:

- Region**: A dropdown menu with the selected value `ap-hongkong`.
- Parameter input method**: Three buttons: `Sheet`, `JSON`, and `Recommended parameters`.
- IdCardType (Optional)**: A dropdown menu with the selected value `HK`.
- NeedVerifyIdCard**: A checkbox that is checked.
- DisableChangeOcrResult (Optional)**: A checkbox that is unchecked.
- DisableCheckOcrWarnings (Optional)**: A checkbox that is unchecked.

The `Online invocation` section is also highlighted with a red box. It includes a `Send request` button and a `Response` section showing the following JSON output:

```
{
  "Response": {
    "RequestId": "e61f9133-e6e0-4f32-9f05-bbcc4b8c3536",
    "SdkToken": "B1289BE4-AD6B-42D8-9A4D-A274B50D115D"
  }
}
```

Region : This is a required parameter that specifies the region information in the domain name and determines the access point. For example, `ap-hongkong` means accessing the Hong Kong access point. APIs may support different regions. For details, see the "Region List" section in the API document of the API you are calling. For example, for [ApplySdkVerificationToken](#), see the "Region List" section in [Common Params](#).

2. Input Parameters

The following request parameter list only provides API request parameters and some common parameters. For the complete common parameter list, see [Common Request Parameters](#).

Parameter Name	Required	Type	Description
Action	Yes	String	Common Params . The value used for the ApplySdkVerificationToken.
Version	Yes	String	Common Params . The value used for the 2018-03-01.
Region	Yes	String	Common Params . For more information see the list of regions supported by the API. This API only supports: ap-hongkong, ap-singapore.
NeedVerifyIdCard	Yes	Boolean	Whether ID card authentication is required. If not, only document OCR will be performed. Currently, authentication is available only when the value of IdCardType is HK .

NeedVerifyIdCard : This is a required parameter that specifies whether to perform identity card authentication. If this parameter is not selected, only document OCR will be performed. You can select this parameter only when the value of **IdCardType** is **HK**.

Step 2.

Select the corresponding backend language, generate the code, and integrate it into your project. Take Java as an example.

1. To integrate the code into your project, you need to import the SDK dependencies (see [Java](#) in the top right corner of the figure below), get key information, and enter the **SecretId** and **SecretKey** in the code for successful calling.
2. Part of the field information in the generated code is subject to the entered content. To adjust an input parameter, modify its value on the left and generate the code again.

Code generating Online invocation Signature generation Parameter description Feedback Data

```
tccli faceid ApplySdkVerificationToken --cli-unfold-argument --region ap-hongkong --IdCardType HK --NeedVerifyIdCard True
```

☒ Java ☐ Python ☐ nodejs ☐ PHP ☐ GO ☐ .net ☐ c++

Download project SDK dependency [i](#) [JAVA SDK Usage Guide](#) [🔗](#)

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.faceid.v20180301.FaceidClient;
import com.tencentcloudapi.faceid.v20180301.models.*;

public class ApplySdkVerificationToken
{
    public static void main(String [] args) {
        try{
            // Required steps:
            // Instantiate an authentication object. The Tencent Cloud account key pair `secretId` and `secretKey` need to
            // be passed in as the input parameters
            // This example uses the way to read from the environment variable, so you need to set these two values in the
            // environment variable in advance
            // You can also write the key pair directly into the code, but be careful not to copy, upload, or share the code
            // with others

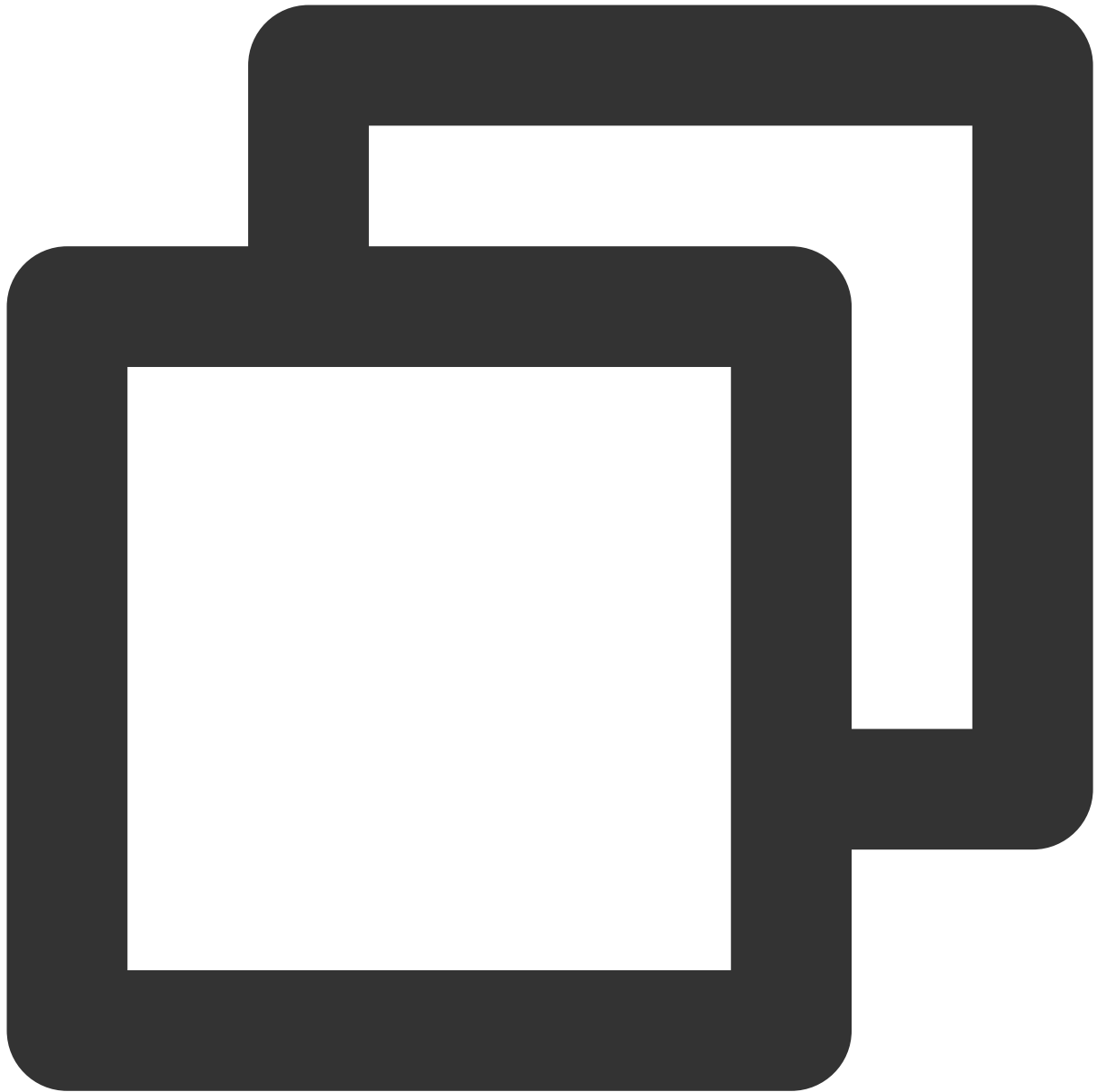
            // Query the CAM key: https://console.tencentcloud.com/capi
            Credential cred = new Credential("SecretId", "SecretKey");
            // Instantiate an HTTP option (optional; skip if there are no special requirements)
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("faceid.tencentcloudapi.com");
            // Optional steps:
            // Instantiate a client configuration object. You can specify the timeout period and other configuration items.
```

Notes

To generate a SecretId and a SecretKey, visit <https://console.tencentcloud.com/cam/capi>.

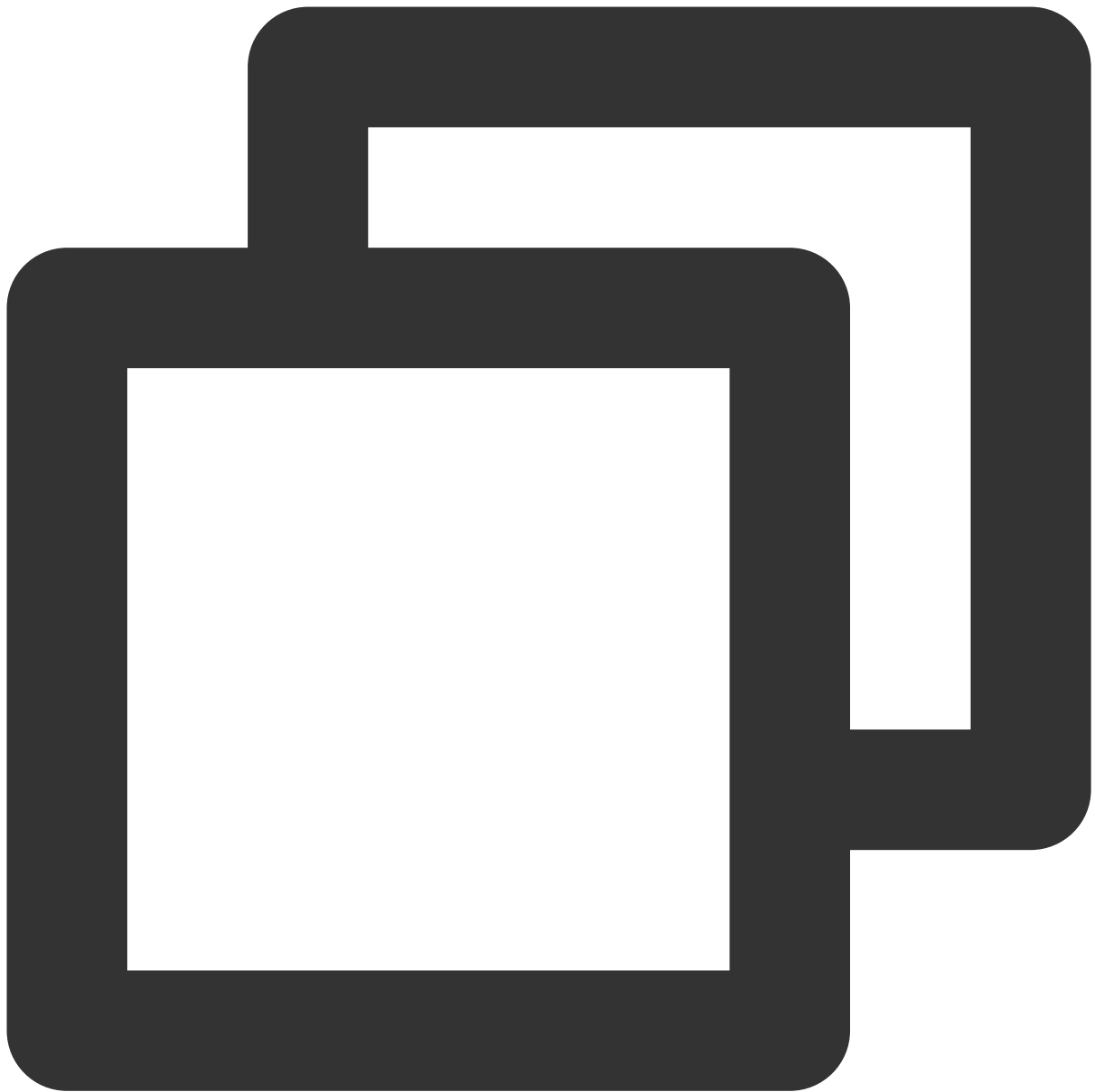
To upload Base64-encoded images or videos for input parameters, remove the `data:image/jpeg;base64,` prefix and the `\\n` line break.

If the following information is returned, manually configure the signature type:



```
[TencentCloudSDKException]message:AuthFailure.SignatureFailure-The provided credent  
could not be validated because of exceeding request size limit, please use new sign  
method `TC3-HMAC-SHA256`. requestId:719970d4-5814-4dd9-9757-a3f11ecc9b20
```

Configuring signature type:



```
`clientProfile.setSignMethod("TC3-HMAC-SHA256"); // Specify the signature algorithm
```

Connecting to TencentCloud API

Last updated : 2023-06-06 16:21:50

TencentCloud API 3.0 integrates SDKs for multiple programming languages, making it easier for you to call APIs. It supports nearby access, and the domain name for nearby access is `faceid.tencentcloudapi.com`. It also supports access using a domain name with a specified region, for example, the domain name for the Singapore region is `faceid.ap-singapore.tencentcloudapi.com`.

We recommend that you use the domain name for nearby access. When you call an API, the request is automatically resolved to a server in the region **nearest** to the location where the API is called. For example, if a request is made in Singapore, it will be automatically resolved to a server in Singapore, which has the same effect as specifying `faceid.ap-singapore.tencentcloudapi.com`.

Note:

1. For latency-sensitive businesses, we recommend that you specify the region in the domain name.
2. A domain name is an API access point and does not represent the region where the product or API actually provides services. For the list of regions supported by the product, see the call method/common parameter document. For the regions supported by the API, see the input parameters part in the API document.

You can select the language you are familiar with for coding. This section uses the Go language as an example to show how to integrate the Tencent Cloud SDK into your server.

[Tencent Cloud SDK 3.0 for Python](#)

[Tencent Cloud SDK 3.0 for Java](#)

[Tencent Cloud SDK 3.0 for PHP](#)

[Tencent Cloud SDK 3.0 for Go](#)

[Tencent Cloud SDK 3.0 for NodeJS](#)

[Tencent Cloud SDK 3.0 for .NET](#)

[Tencent Cloud SDK 3.0 for C++](#)

Environmental dependency

1. Go 1.9 or above, with the necessary environment variables such as `GOPATH` set properly
2. Activate the eKYC product by following the [process guide](#)
3. Obtain SecretId and SecretKey by following the [guide to getting the secret key](#)

Installation

Installing through go get (recommended)

We recommend you install the SDK by using the tool that comes with the language.



```
go get -u github.com/tencentcloud/tencentcloud-sdk-go-intl-en
```

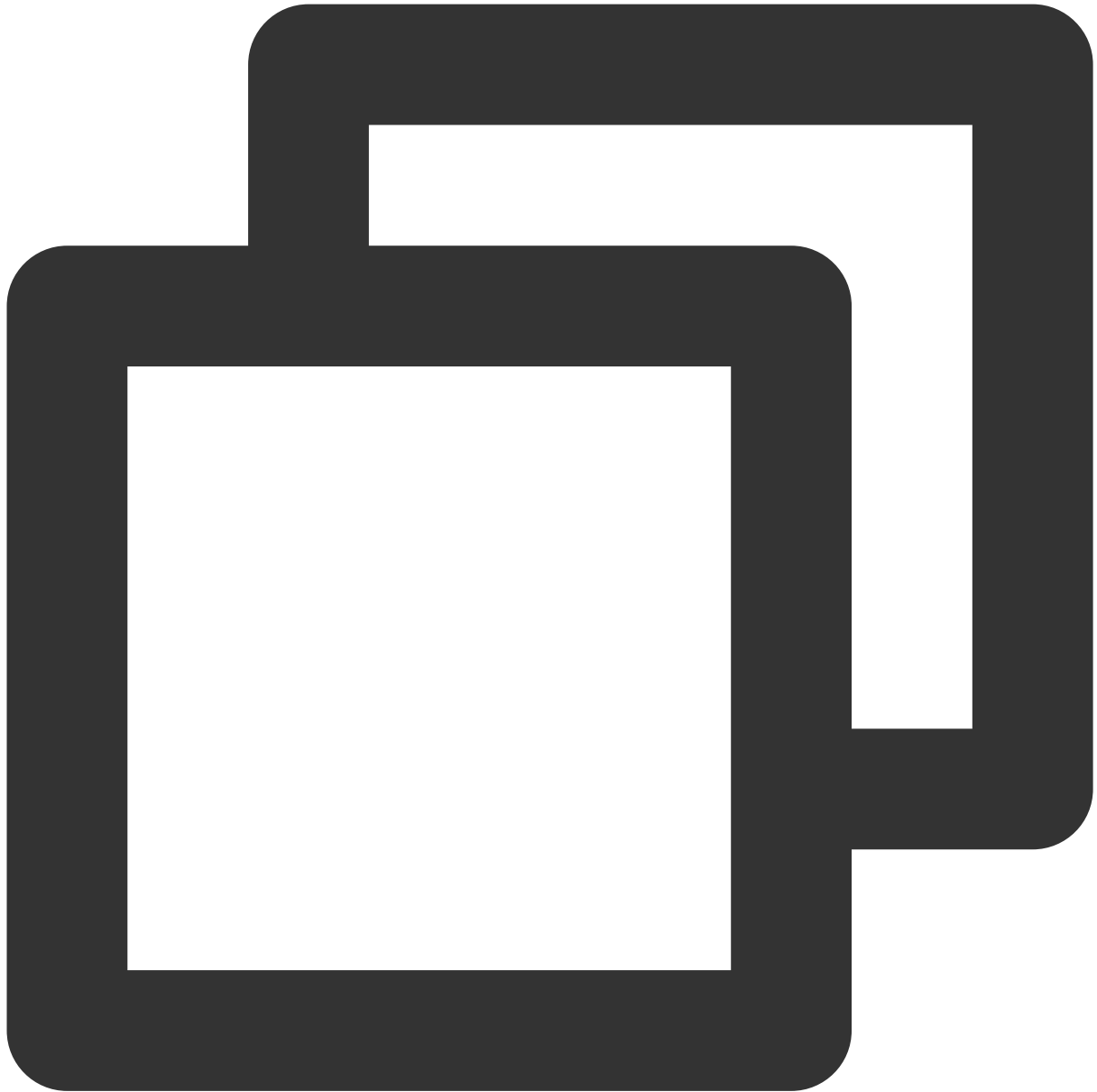
Installing through source code

Go to the [GitHub](#) page to download the latest code, and decompress it to

```
$GOPATH/src/github.com/tencentcloud .
```

Integrating into the server

After the Tencent Cloud SDK is installed, you can use the import command to integrate the SDK into the server. The sample code is as follows:



```
package main

import (
    "fmt"
    cloud "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/common/profile"
    faceid "github.com/tencentcloud/tencentcloud-sdk-go-intl-en/tencentcloud/faceid/"
)
```

```
func main() {  
    // Instantiate a client configuration object. You can specify the timeout period  
    prof := profile.NewClientProfile()  
    prof.HttpProfile.ReqTimeout = 60  
    // TODO replace the SecretId and SecretKey string with the API SecretId and SecretKey  
    credential := cloud.NewCredential("SecretId", "SecretKey")  
    // Instantiate the client object of the requested faceid  
    FaceIdClient, _ := faceid.NewClient(credential, "ap-singapore", prof)  
  
    // Instantiate the request object and provide necessary parameters  
    request := faceid.NewGetFaceIdTokenIntlRequest()  
    var SecureLevel = "4"  
    request.SecureLevel = &SecureLevel  
    // Call the Tencent Cloud API through FaceIdClient  
    response, _ := FaceIdClient.GetFaceIdTokenIntl(request)  
    // Process the Tencent Cloud API response  
    fmt.Println("response: ", *response.Response.SdkToken)  
}
```

Error Codes

Last updated : 2023-07-04 16:31:45

This document describes all error codes of the eKYC product and whether a request is billed ("N" for no and "Y" for yes) in the case of an error code.

Common Error Codes (for All Services)

Error Code	Description	Billed
InvalidParameter	Invalid parameter	N
InvalidParameterValue.BizTokenIllegal	Invalid BizToken	N
InvalidParameterValue.BizTokenExpired	BizToken expired	N
UnauthorizedOperation.Nonactivated	The service has not been activated.	N
UnauthorizedOperation.Activating	The service is being activated	N
UnauthorizedOperation.ActivateError	The service has not been activated	N
UnauthorizedOperation.Arrears	The account has overdue payments	N
OperationDenied	Operation denied	N
UnauthorizedOperation.NonAuthorize	Identity verification has not been completed for the account.	N
UnauthorizedOperation	Unauthorized operation	N
FailedOperation.ImageSizeTooLarge	Image size too large	N
InternalError	Internal error	N

The following tables list the error codes of the eKYC subproducts:

Liveness Detection and Face Comparison (Mobile HTML5) Error Codes

--	--	--

Error Code	Description	Billed
0	Success	Y
1	Invalid parameter	N
2	Incorrect parameter value	N
3	Bad request	N
4	Internal service error	N
8	Internal service error	N
9	Internal service error	N
10	Internal service error	N
11	Internal service error	N
14	Verification has been completed	N
15	Token expired	N
16	Retry limit has been reached	N
202	Failed to get video	N
1001	Failed to call the liveness engine	N
1002	Suspected spoofed recording.	N
1003	Real person detection failed	N
1004	Face detection failed	N
1005	Liveness detection failed	N
1101	No sound is detected	N
1102	The face is not fully exposed	N
1103	Speech recognition failed	N
1104	The video format is incorrect	N
1105	Failed to pull the video. Please try again	N
1106	The volume of the video is too low	N

1107	The video is empty or its size is inappropriate. The recording duration should be about 6 seconds	N
1108	The video definition is too low	N
1109	The lip movement range is too small	N
1201	The lighting is too dim	N
1202	The lighting is too strong	N
1203	The face is too close to the screen	N
1204	The face is too far right from the screen	N
1205	The face is too far from the screen	N
1206	The face is too far left from the screen	N
1207	No motions of eye closing are detected	N
1208	The first motion is not detected	N
1209	No motions of mouth opening are detected	N
1210	Failed to detect a full face	N
1301	Real person detection failed	N
1302	Real person detection did not reach the passing standard	N
1303	The video is too short. Please capture a video longer than 2 seconds	N
2001	Error calling the comparison engine	N
2004	The image passed in is too large or too small	N
2010	Multiple faces are detected	N
2011	Real person comparison failed	N
2012	Failed to detect a full face	N
2013	No faces are detected	N
2014	The resolution of the image passed in is too low. Please upload a new one	N
2015	Comparison failed	N
2016	The comparison similarity did not reach the passing standard	Y

1401	face reflect liveness detect failed	N
------	-------------------------------------	---

Liveness Detection and Face Comparison (App SDK) Error Codes

Error Code	Description	Billed
0	Succeeded	Y
1001	System error	N
1004	Liveness detection and face comparison failed	N
2004	The image passed in is too large or too small	N
2012	Several faces were detected	N
2013	No face was detected, or the face detected was incomplete	N
2014	The image resolution is too low or the quality does not meet the requirements	N
2015	Face comparison failed	N
2016	The similarity did not reach the standard passing threshold	Y
-999	The verification process wasn't finished	N

Liveness Detection and Face Comparison (Pure API) Error Codes

Error Code	Description	Billed
Success	Success	Y
FailedOperation.CompareLowSimilarity	The comparison similarity did not reach the passing standard.	Y
FailedOperation.ActionCloseEye	No motions of eye closing are detected.	N
FailedOperation.ActionFaceClose	The face is too close to the screen.	N
FailedOperation.ActionFaceFar	The face is too far from the screen.	N
FailedOperation.ActionFaceLeft	The face is too far left from the screen.	N

FailedOperation.ActionFaceRight	The face is too far right from the screen.	N
FailedOperation.ActionFirstAction	No movement is detected.	N
FailedOperation.ActionLightDark	The lighting is too dim.	N
FailedOperation.ActionLightStrong	The lighting is too strong.	N
FailedOperation.ActionNodetectFace	Failed to detect a full face.	N
FailedOperation.ActionOpenMouth	No motions of mouth opening are detected.	N
FailedOperation.CompareFail	Comparison failed.	N
FailedOperation.CompareSystemError	Error calling the comparison engine API.	N
FailedOperation.DownLoadError	File download failed.	N
FailedOperation.DownLoadTimeoutError	File download timed out.	N
FailedOperation.LifePhotoDetectFaces	Multiple faces are detected.	N
FailedOperation.LifePhotoDetectFake	Real person comparison failed.	N
FailedOperation.LifePhotoDetectNoFaces	Failed to detect a full face.	N
FailedOperation.LifePhotoPoorQuality	The resolution of the image passed in is too low. Please upload a new one.	N
FailedOperation.LifePhotoSizeError	The image passed in is too large or too small.	N
FailedOperation.LipFaceIncomplete	The face is not fully exposed.	N
FailedOperation.LipMoveSmall	The lip movement range is too small.	N
FailedOperation.LipNetFailed	Failed to pull the video. Please try again.	N
FailedOperation.LipSizeError	The video is empty or its size is inappropriate. The recording duration should be about 6 seconds.	N
FailedOperation.LipVideoInvalid	The video format is incorrect.	N
FailedOperation.LipVideoQuaility	The video definition is too low.	N
FailedOperation.LipVoiceDetect	No sound is detected.	N
FailedOperation.LipVoiceLow	The volume of the video is too low.	N
FailedOperation.LipVoiceRecognize	Speech recognition failed.	N

FailedOperation.LivessBestFrameError	Face detection failed. Unable to extract the photo for comparison.	N
FailedOperation.LivessDetectFail	Liveness detection failed.	N
FailedOperation.LivessDetectFake	Suspected spoofed recording.	N
FailedOperation.LivessSystemError	Error calling the liveness engine API.	N
FailedOperation.LivessUnknownError	Video-based real person detection failed.	N
FailedOperation.SilentDetectFail	Real person detection failed.	N
FailedOperation.SilentEyeLiveFail	Eye detection failed.	N
FailedOperation.SilentFaceDetectFail	No face is detected in the video.	N
FailedOperation.SilentFaceQualityFail	Low face quality.	N
FailedOperation.SilentFaceWithMaskFail	A face mask is detected.	N
FailedOperation.SilentMouthLiveFail	Mouth detection failed.	N
FailedOperation.SilentMultiFaceFail	Multiple faces are detected in the video.	N
FailedOperation.SilentPictureLiveFail	The video might be spoofed.	N
FailedOperation.SilentThreshold	Real person detection did not reach the passing standard.	N
FailedOperation.SilentTooShort	The video is too short. Please capture a video longer than 2 seconds.	N
FailedOperation.UnKnown	Unknown internal error.	N
InvalidParameter	Invalid parameter.	N
InvalidParameterValue	Incorrect parameter value.	N
UnauthorizedOperation	Unauthorized operation.	N
UnauthorizedOperation.Arrears	The account has overdue payments	N
UnauthorizedOperation.NonAuthorize	Identity verification has not been completed for the account.	N
UnauthorizedOperation.Nonactivated	The service has not been activated.	N
UnsupportedOperation	Unsupported operation.	N

Identity Verification (App SDK) Error Codes

Error Code	Description	Billed
0	Success	Y
1001	Failed to call the liveness engine	Y
1004	Face detection failed	Y
2004	The image passed in is too large or too small	Y
2012	Failed to detect a full face	Y
2013	No faces are detected	Y
2014	The resolution of the image passed in is too low. Please upload a new one	Y
2015	Comparison failed	Y
2016	The comparison similarity did not reach the passing standard	Y