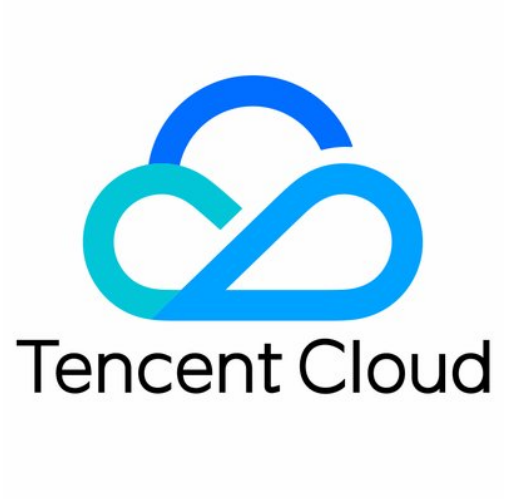
Tencent Effect SDK

SDK統合ガイド

製品ドキュメント



カタログ：

SDK統合ガイド
Tencent Effect SDKの独立した統合
ios

最終更新日：：2023-02-27 14:18:15

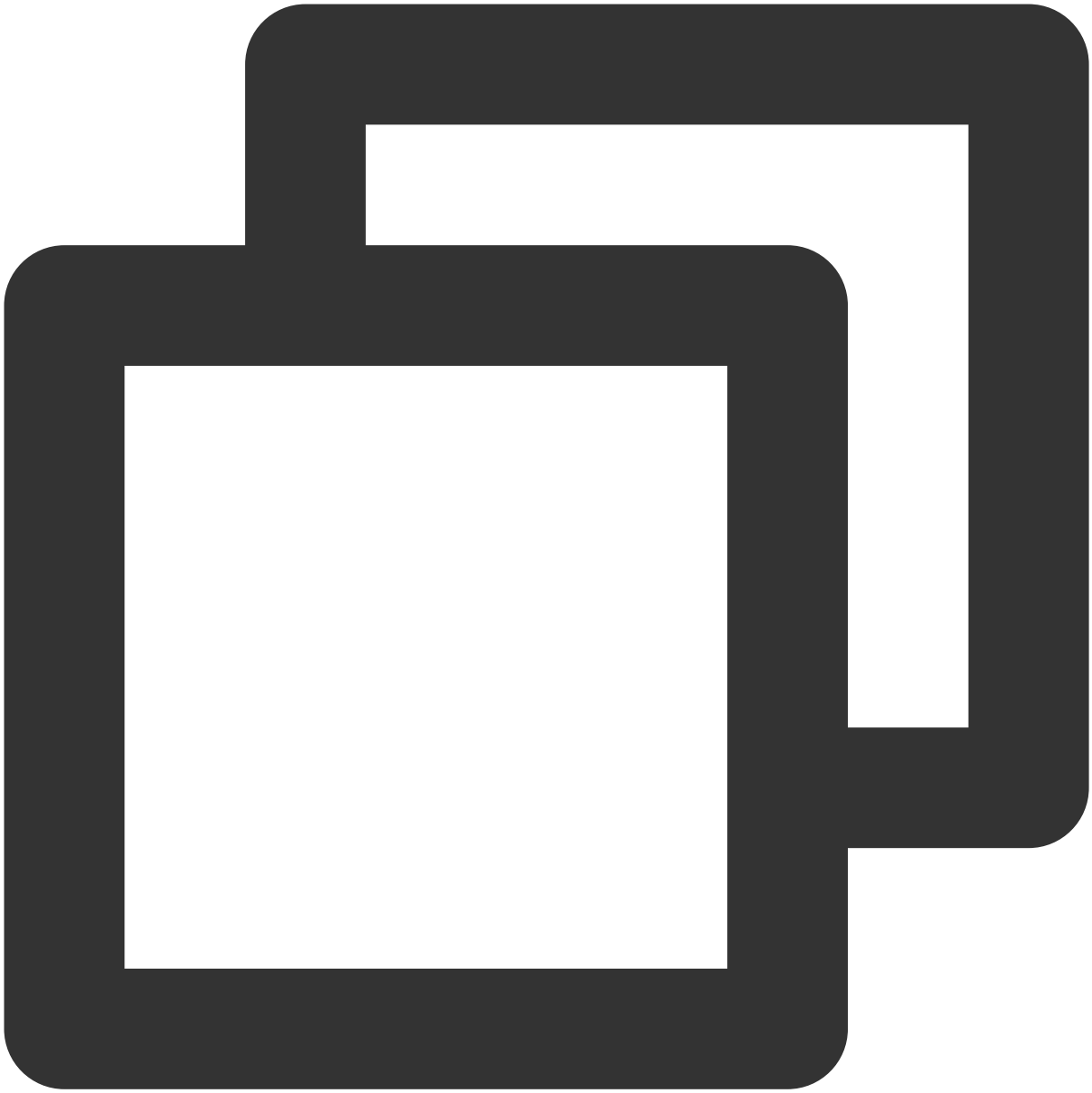
統合の準備

開発者環境要件

開発ツールXCode 11およびそれ以上：App StoreまたはダウンロードアドレスをクリックしますClick。
推奨実行環境：
デバイス要件：iPhone 5およびそれ以上である必要があります。iPhone 6およびそれ未満の場合は、フロントカメラのサポートを最大720pとし、1080pはサポートしていません。
システム要件：iOS 10.0およびそれ以降のバージョン。

SDKのインポート

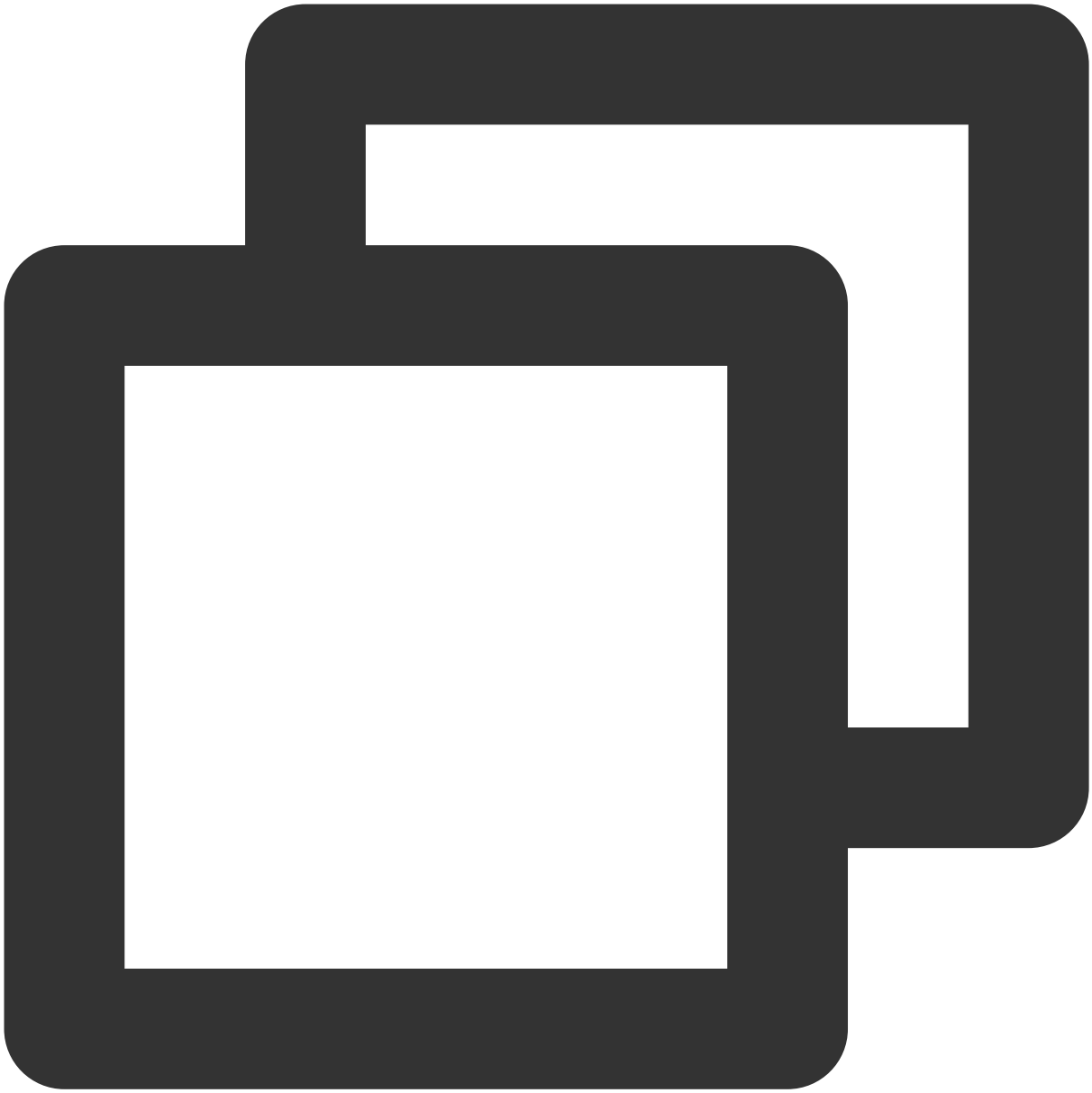
CocoaPodsソリューションを使用するか、またはまずSDKをローカルにダウンロードしてから手動で現在のプロジェクトにインポートするかを選択できます。
CocoaPodsの使用
SDKのダウンロードと手動でのインポート
動的ダウンロードと統合
1. **CocoaPodsのインストール**
端末のウィンドウに次のコマンドを入力します（事前にMac にRuby環境をインストールしておく必要があります）。



```
sudo gem install cocoapods
```

2. **Podfileファイルの新規作成**

プロジェクトが存在するパスに入り、次のコマンドラインを入力するとプロジェクトパスの下にPodfile ファイルが現れます。



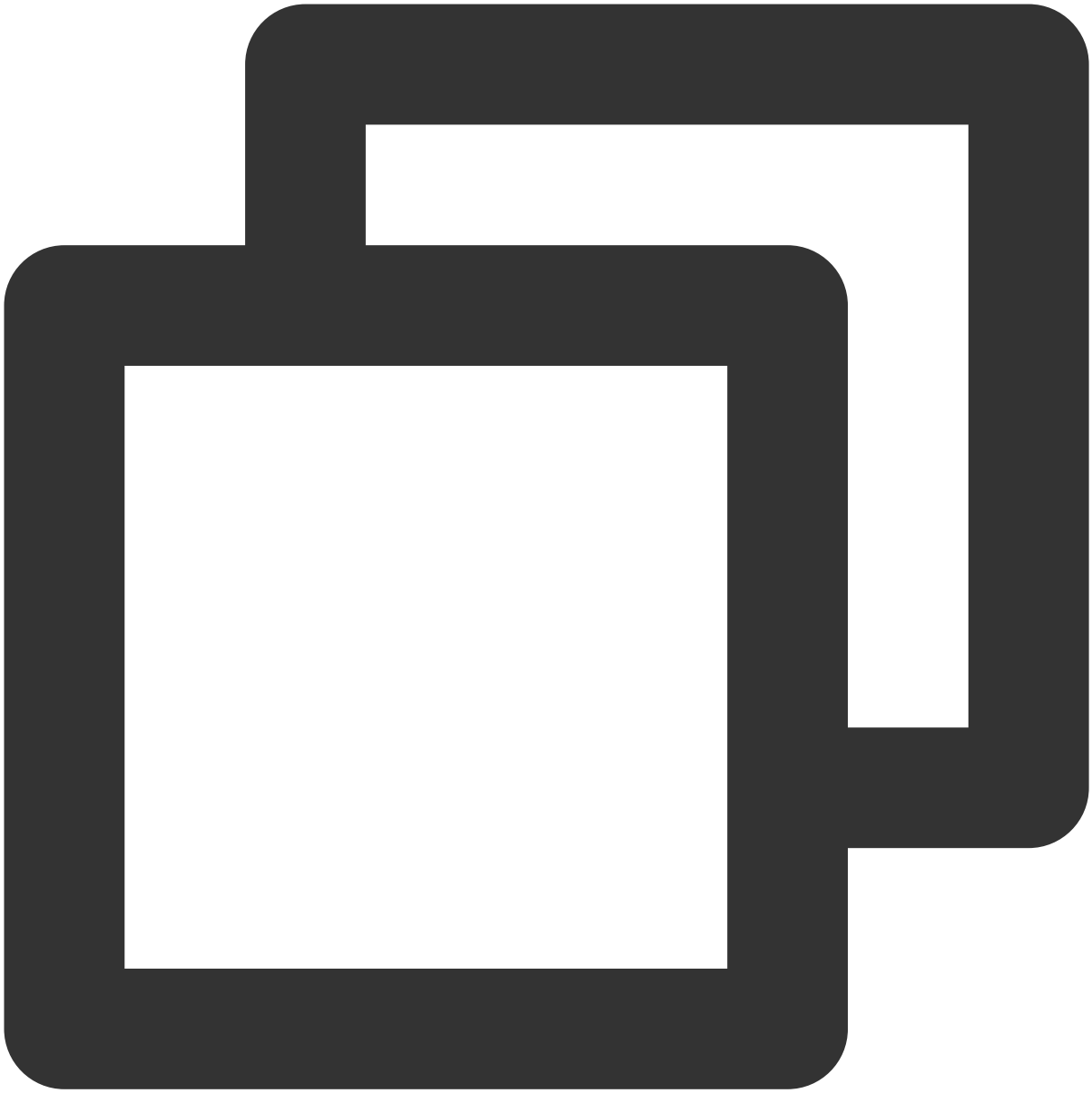
```
pod init
```

3. **Podfileファイルの編集**

プロジェクトのニーズに応じて適切なバージョンを選択し、Podfileを編集します：

XMagicスタンダード版

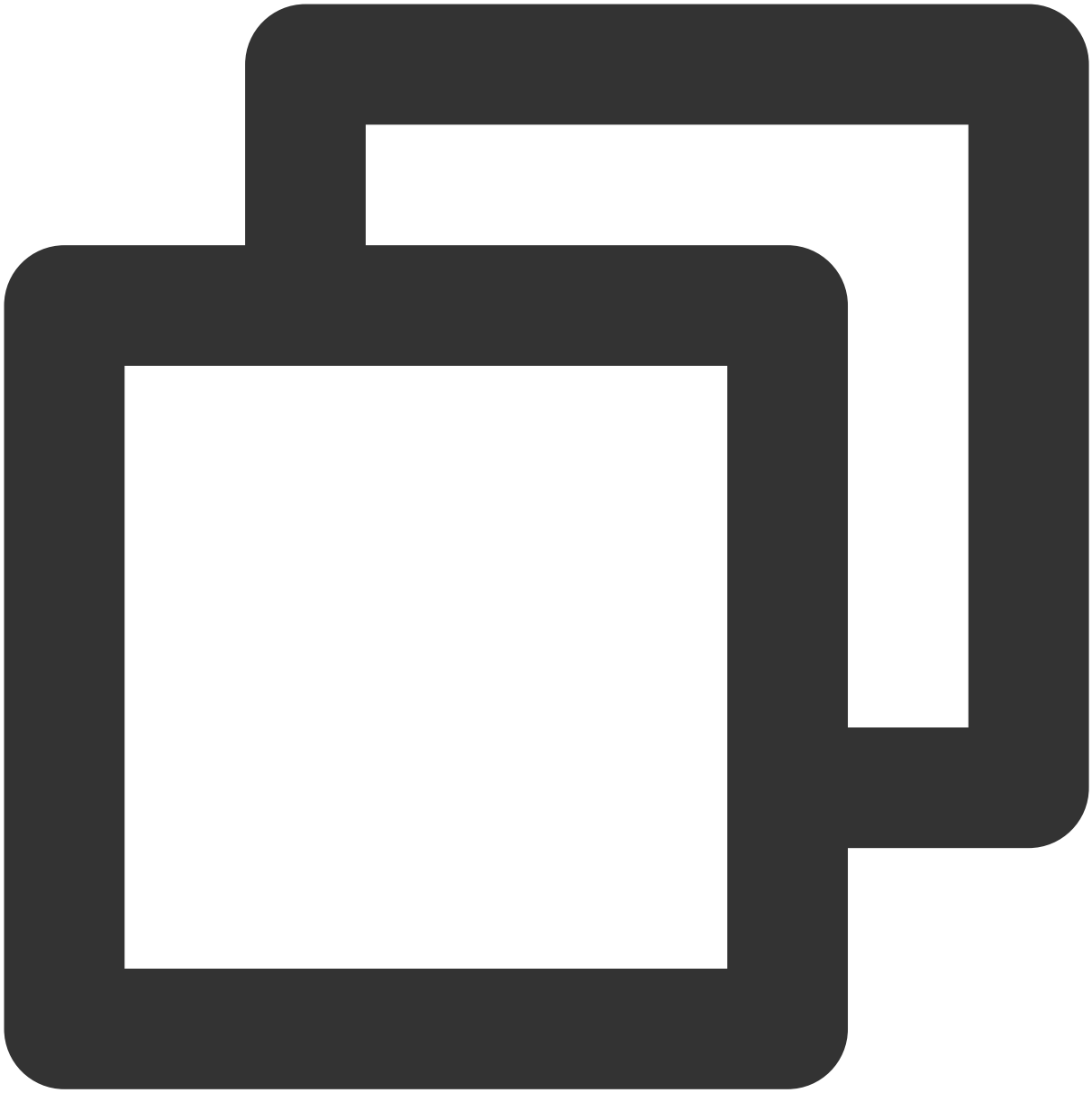
次の方法でPodfileファイルの編集を行ってください。



```
platform :ios, '8.0'

target 'App' do
pod 'XMagic'
end
```

XMagic簡易版

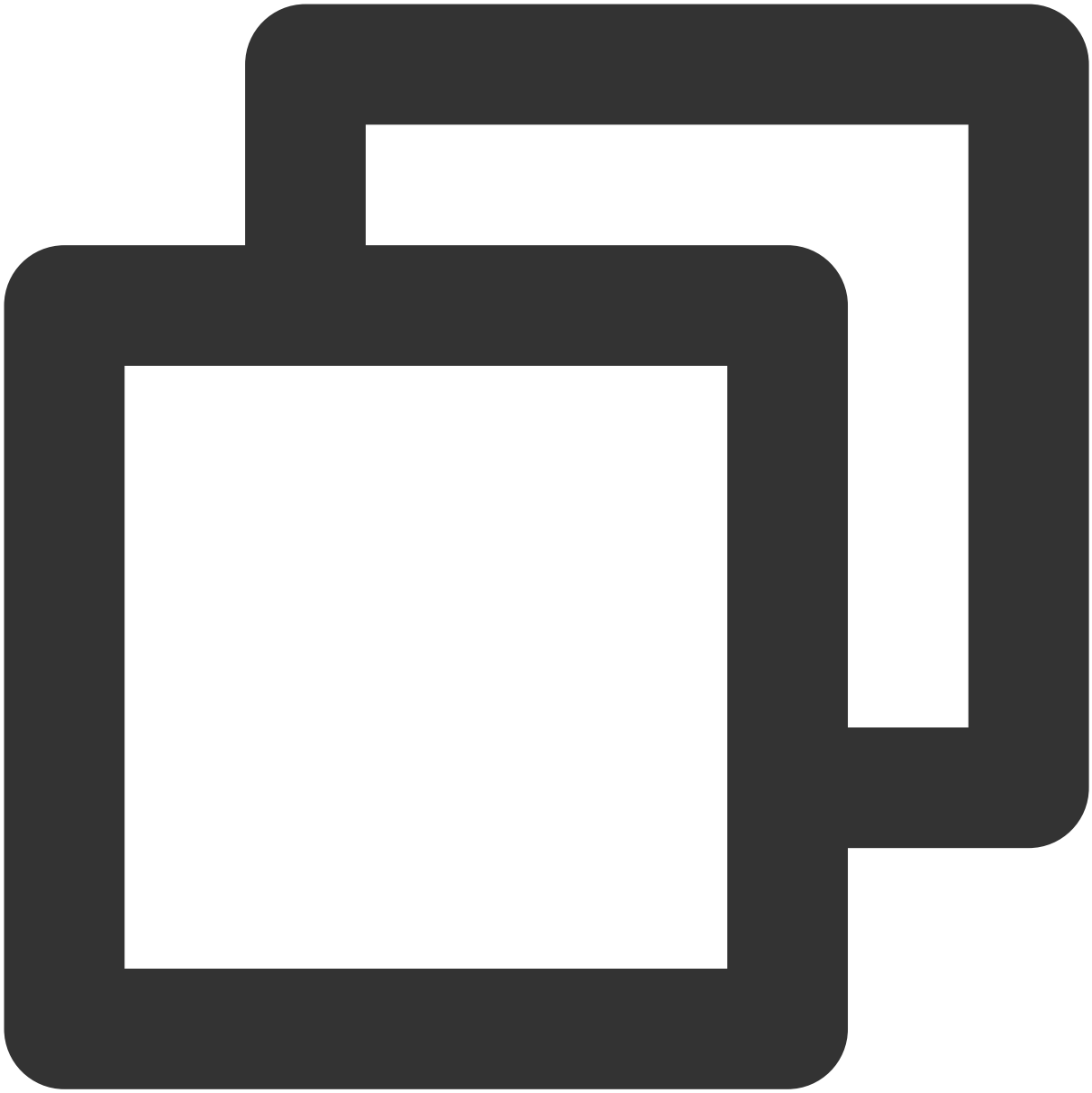
インストールパッケージのボリュームがスタンダード版より小さく、ベーシック版A1-00、ベーシック版A1-01、プレミアム版S1-00のみをサポートしています。次の方法でPodfileファイルの編集を行ってください。



```
platform :ios, '8.0'

target 'App' do
pod 'XMagic_Smart'
end
```

4. **SDKの更新およびインストール**

端末ウィンドウに次のコマンドを入力し、ローカルライブラリファイルを更新し、SDKをインストールします。



```
pod install
```

podコマンドが実行されると、SDKに統合された `.xcworkspace` サフィックスが付いたプロジェクトファイルが作成され、ダブルクリックして開くことができます。

5. **実際のプロジェクトへの美顔リソースの追加**

対応するパッケージのSDKおよび美顔リソースをダウンロードして解凍し、**resources**フォルダ下の**LightCore.bundle**、**Light3DPlugin.bundle**、**LightBodyPlugin.bundle**、**LightHandPlugin.bundle**、**LightSegmentPlugin.bundle**、**audio2exp.bundle**以外の**その他のbundleリソース**を実際のプロジェクトに追加します。

Build SettingsのOther Linker Flagsに `-ObjC` を追加します。

6. Bundle Identifierを、テスト用に申請した権限と同じものに変更します。

1. SDKおよび美顔リソースをダウンロードして解凍します。frameworksフォルダ内にあるのがsdk、resourcesフォルダ内にあるのが美顔のbundleリソースです。

2. **SDKのバージョンが2.5.1より古い場合：**

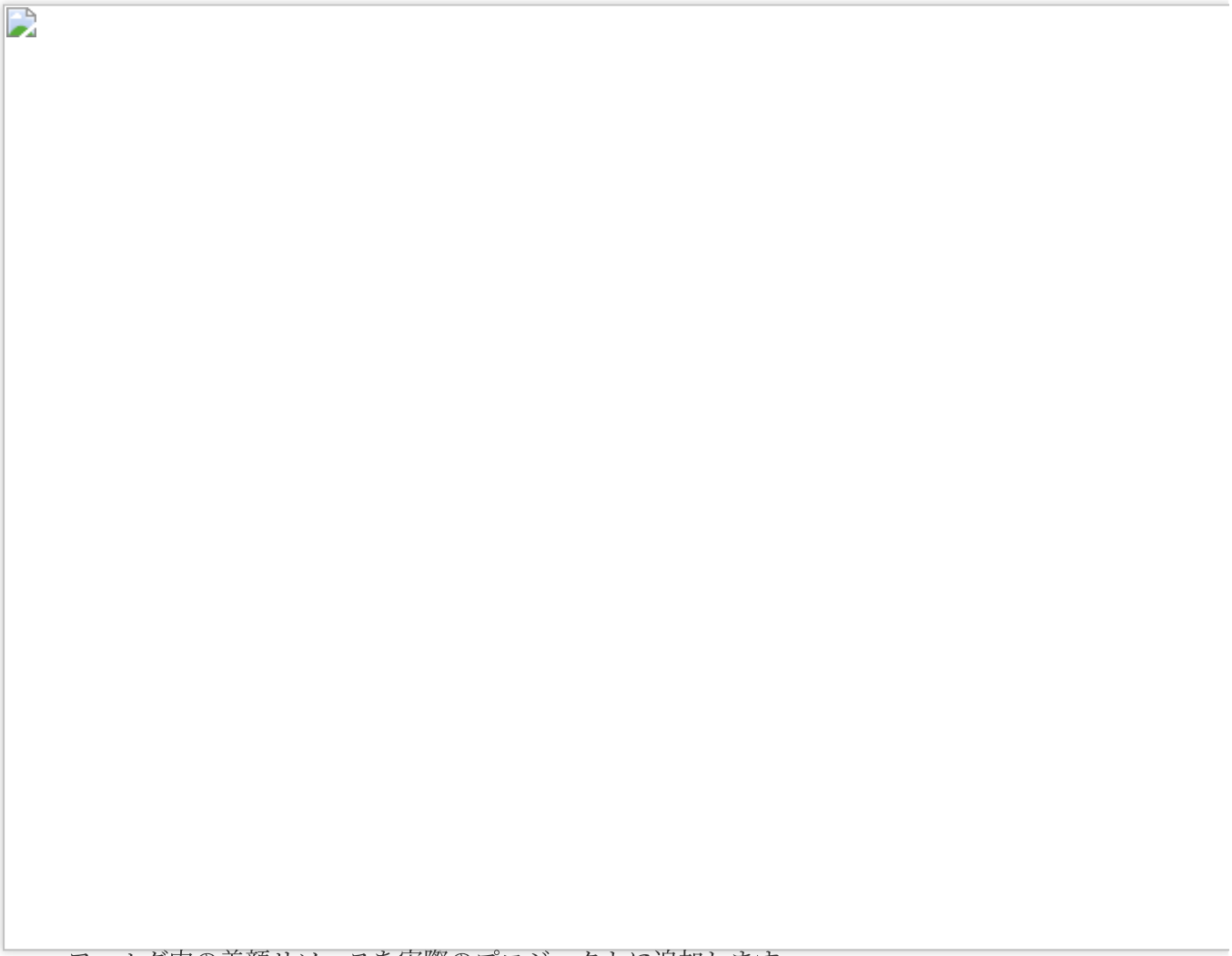
ご自身のXcodeプロジェクトを開き、frameworksフォルダ内のframeworkを実際のプロジェクトに追加します。実行したいtargetを選択し、**General**の項目を選び、**Frameworks,Libraries,and Embedded Content**の項目をクリックして展開し、下部の「+」アイコンをクリックして依存ライブラリを追加します。ダウンロードした `XMagic.framework` 、 `YTCommonXMagic.framework` 、 `libpag.framework` およびそれらに必要な依存ライブラリ `MetalPerformanceShaders.framework` 、 `CoreTelephony.framework` 、 `JavaScriptCore.framework` 、 `VideoToolbox.framework` 、 `libc++.tbd` を順に追加します。必要に応じてその他のツールライブラリ、 `Masonry.framework` （ウィジェットレイアウトライブラリ）、 `SSZipArchive` （ファイル解凍ライブラリ）を追加します。

SDKのバージョンが2.5.1およびそれ以降の場合：

Xcodeプロジェクトを開き、frameworksフォルダ内のframeworkを実際のプロジェクトに追加し、動作させたいtargetを選択し、**General**項目を選び、

Frameworks,Libraries,and Embedded Contentの項目をクリックして展開し、下部の「+」アイコンをクリックして依存ライブラリを追加します。ダウンロードした `XMagic.framework` 、

`YTCommonXMagic.framework` 、 `libpag.framework` 、 `Audio2Exp.framework` 、 `TEFFmpeg.framework` およびそれらに必要な依存ライブラリ `MetalPerformanceShaders.framework` 、 `CoreTelephony.framework` 、 `JavaScriptCore.framework` 、 `VideoToolbox.framework` 、 `libc++.tbd` を順に追加し、必要に応じてその他のツールライブラリ、 `Masonry.framework` （ウィジェットレイアウトライブラリ）、 `SSZipArchive` （ファイル解凍ライブラリ）を追加します。



3. resourcesフォルダ内の美顔リソースを実際のプロジェクトに追加します。
4. Build SettingsのOther Linker Flagsに `-ObjC` を追加します。
5. Bundle Identifierを、テスト用に申請した権限と同じものに変更します。

パッケージのサイズを小さくするため、SDKに必要なモデルリソースおよび動的エフェクトリソースMotionRes（一部のベーシック版SDKには動的エフェクトリソースはありません）をネットワークからのダウンロードに変更することができます。ダウンロード成功後、上記ファイルのパスをSDKに設定します。
Demoのダウンロードロジックを再利用することをお勧めします。もちろん、既存のダウンロードサービスを使用することもできます。動的ダウンロードの詳細なガイドについては、SDKパッケージのスリム化（iOS）をご参照ください。

権限の設定

info.plistファイルに対応する権限の説明を追加します。これを行わなければ、iOS 10システム上でプログラムがクラッシュする場合があります。Privacy - Camera Usage Descriptionでカメラの権限を有効にし、Appによるカメラの使用を許可してください。

統合の手順

ステップ1：認証

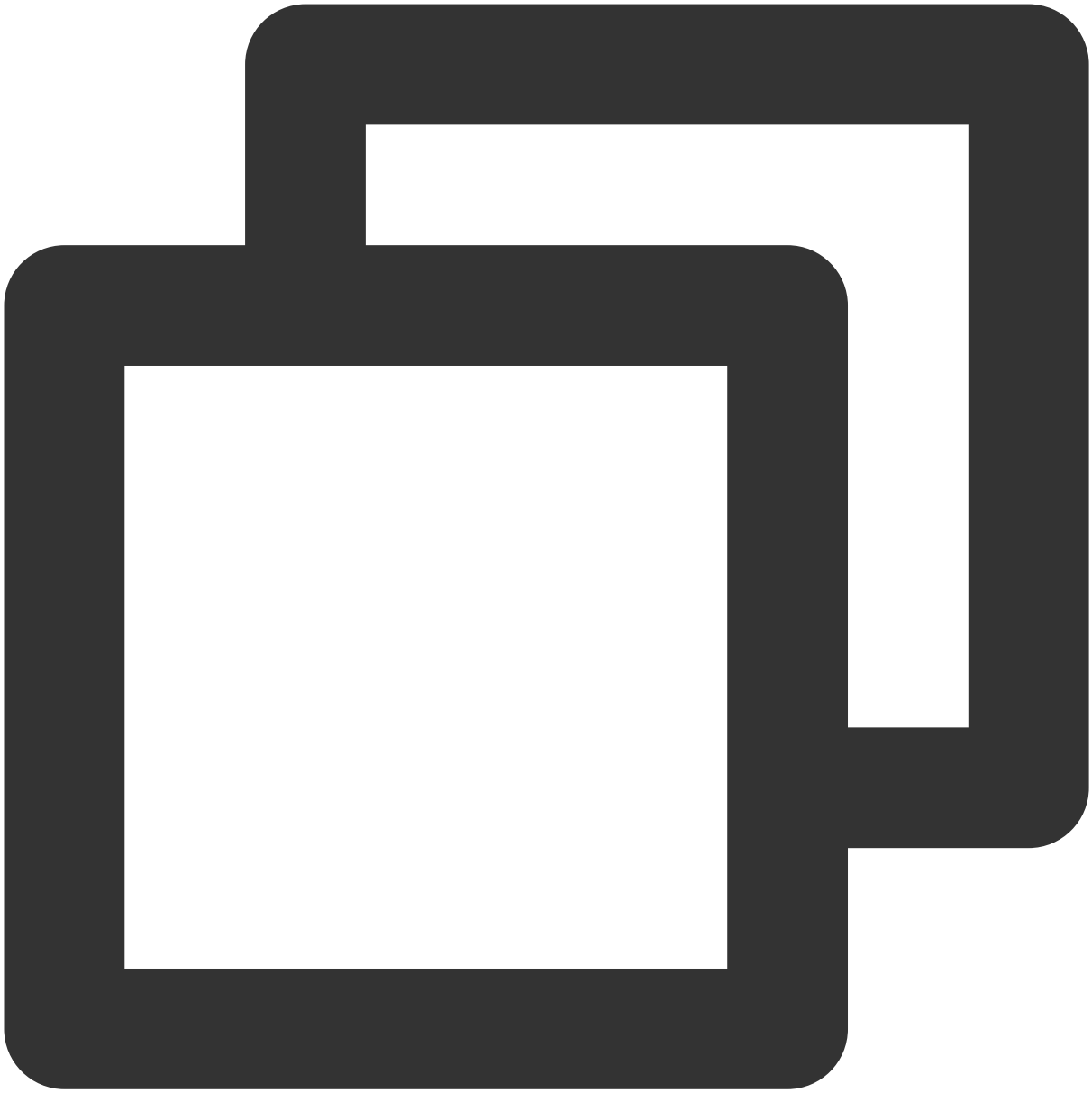
1. 権限承認を申請し、LicenseURLとLicenseKEYを取得します。

注意

正常な状況では、Appのネットワーク接続が一度成功すれば認証フローは完了するため、Licenseファイルをプロジェクトのプロジェクトディレクトリに保存する**必要はありません**。ただし、Appがネットワークに未接続の状態でSDKの関連機能を使用する必要がある場合は、Licenseファイルをダウンロードしてプロジェクトディレクトリに保存し、最低保証プランとすることができます。この場合、Licenseファイル名は必ず `v_cube.license` としなければなりません。

2. 関連業務モジュールの初期化コードの中でURLとKEYを設定し、licenseのダウンロードをトリガーします。使用する直前になってダウンロードすることは避けてください。あるいはAppDelegateのdidFinishLaunchingWithOptionsメソッドでダウンロードをトリガーすることもできます。このうち、LicenseURLとLicenseKeyはコンソールでLicenseをバインドした際に生成された権限承認情報です。

SDKのバージョンが2.5.1より古い場合、 `TELicenseCheck.h` は `XMagic.framework` 内にあり、SDKのバージョンが2.5.1およびそれ以降の場合、 `TELicenseCheck.h` は `YTCommonXMagic.framework` 内にあります。



```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authr
 if (authresult == TELicenseCheckOk) {
     NSLog(@"認証成功");
 } else {
     NSLog(@"認証失敗");
 }
}];
```

認証errorCode説明：

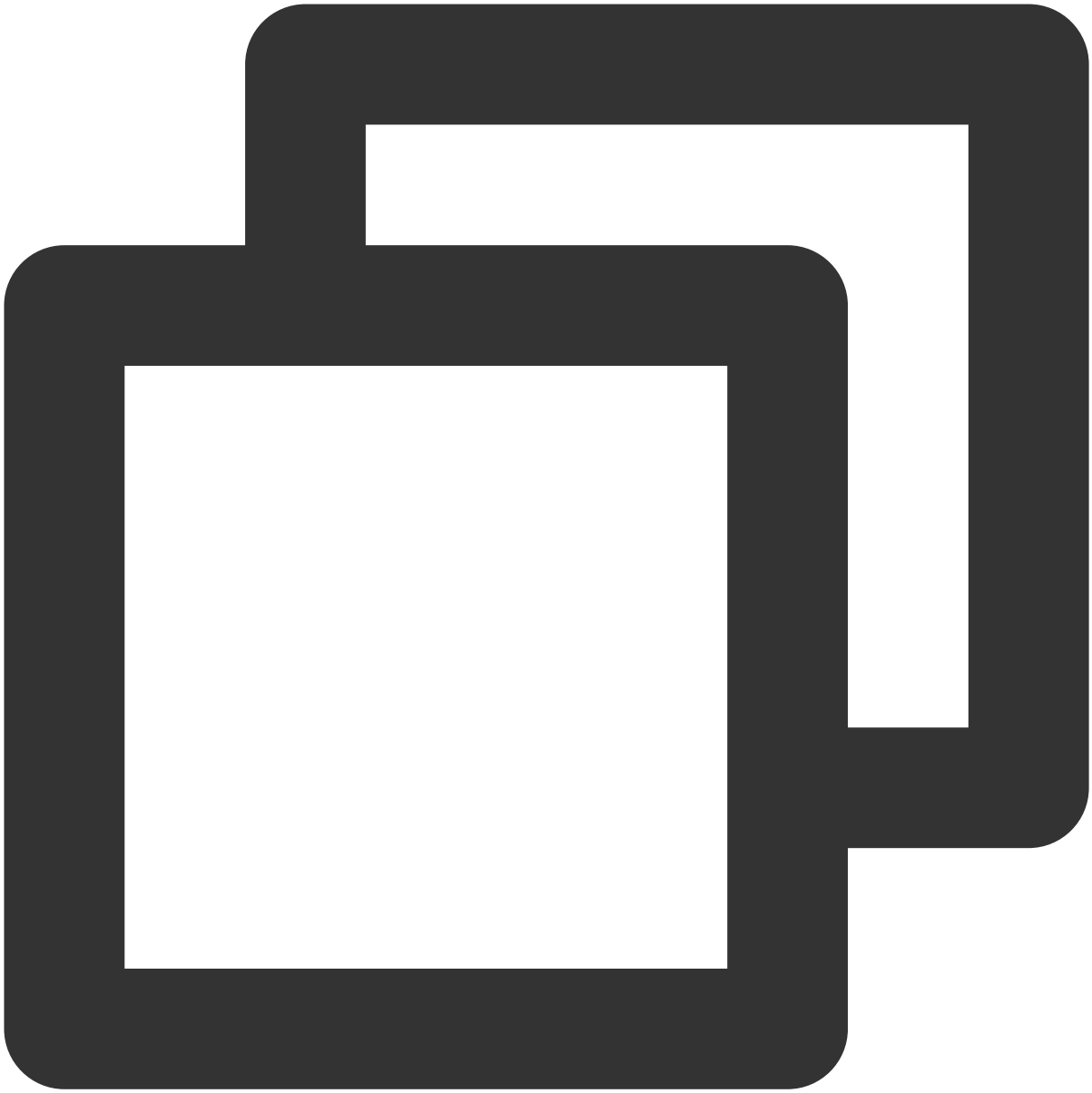
エラーコード	説明

0	成功です。Success
-1	入力パラメータが無効です（例：URLまたはKEYが空など）
-3	ダウンロードの段階で失敗しました。ネットワークの設定を確認してください
-4	ローカルから読み取ったTE権限承認情報が空です。IOの失敗による可能性があります
-5	読み取ったVCUBE TEMP Licenseファイルの内容が空です。IOの失敗による可能性があります
-6	v_cube.licenseファイルのJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-7	署名の検証に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-8	復号に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-9	TELicenseフィールド内のJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-10	ネットワークから解析したTE権限承認情報が空です。Tencent Cloudチームに連絡して処理を依頼してください
-11	TE権限承認情報をローカルファイルに書き込む際に失敗しました。IOの失敗による可能性があります
-12	ダウンロードに失敗しました。ローカルassetの解析も失敗しました
-13	認証に失敗しました
その他	Tencent Cloudチームに連絡して処理を依頼してください

ステップ2：SDKのロード（XMagic.framework）

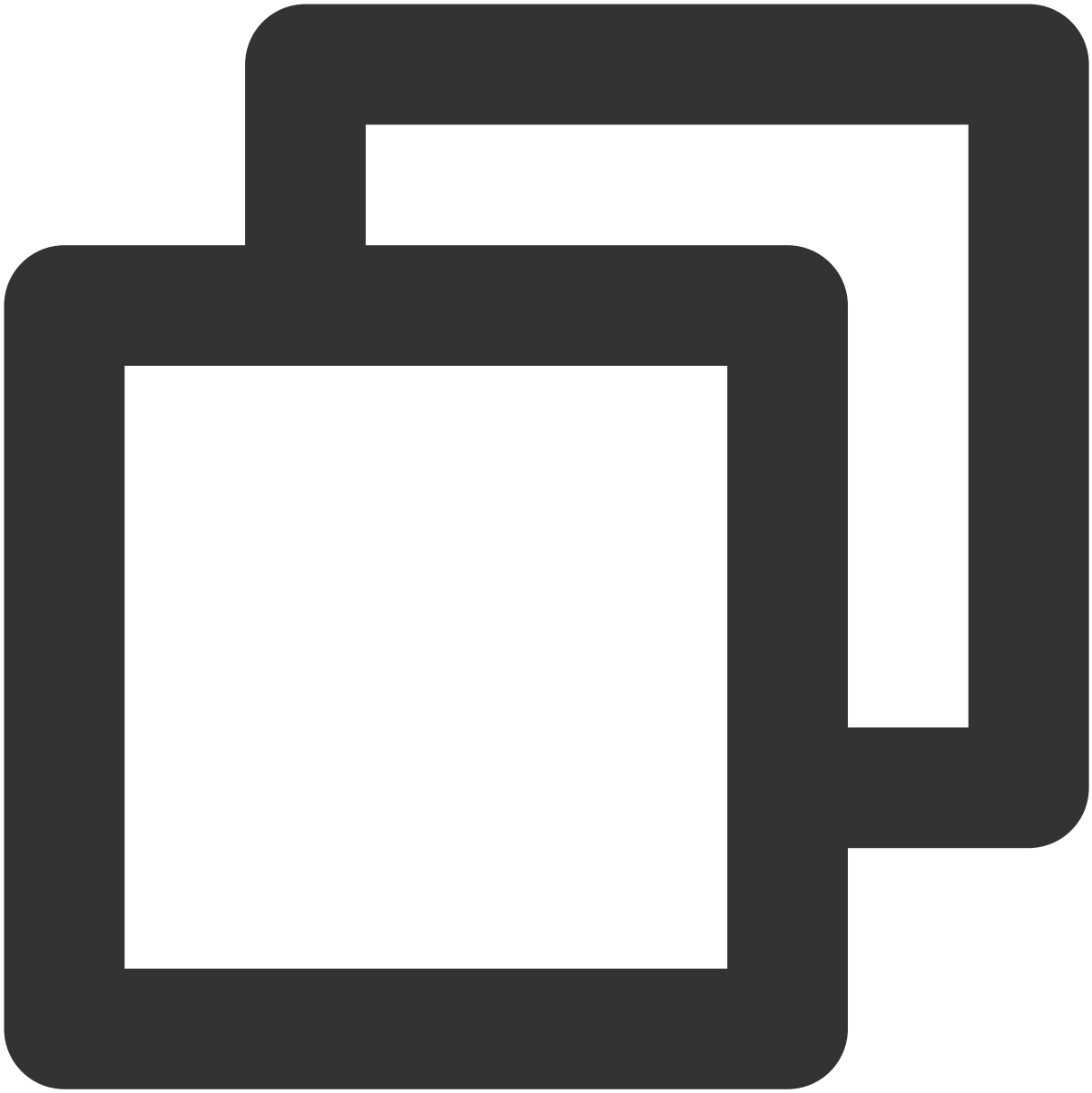
Tencent Effect SDK使用のライフサイクルはおおむね次のとおりです。

1. 美顔関連リソースをロードします。



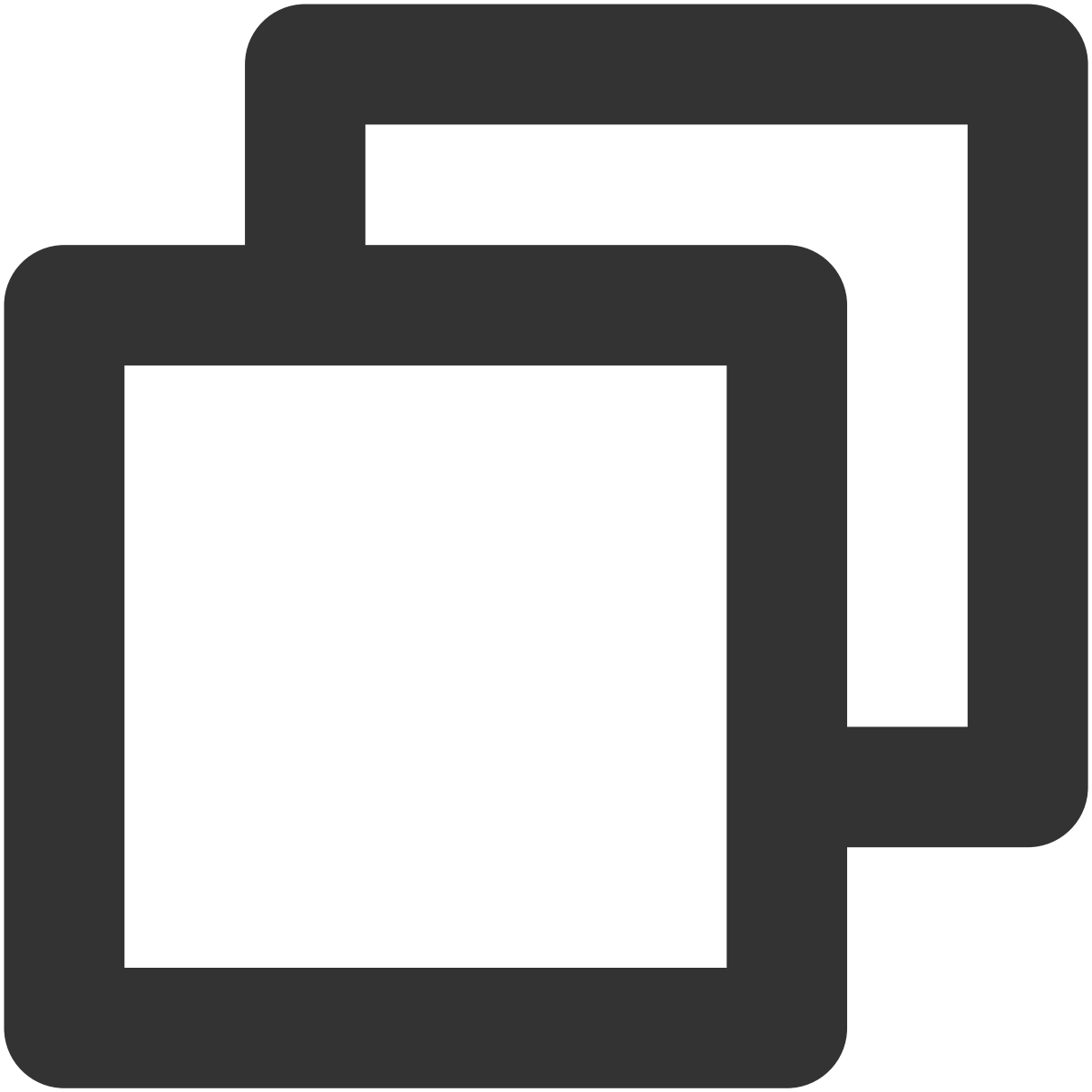
```
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
 @"root_path":[[NSBundle mainBundle] bundlePath]
};
```

2. Tencent Effect SDKを初期化します。

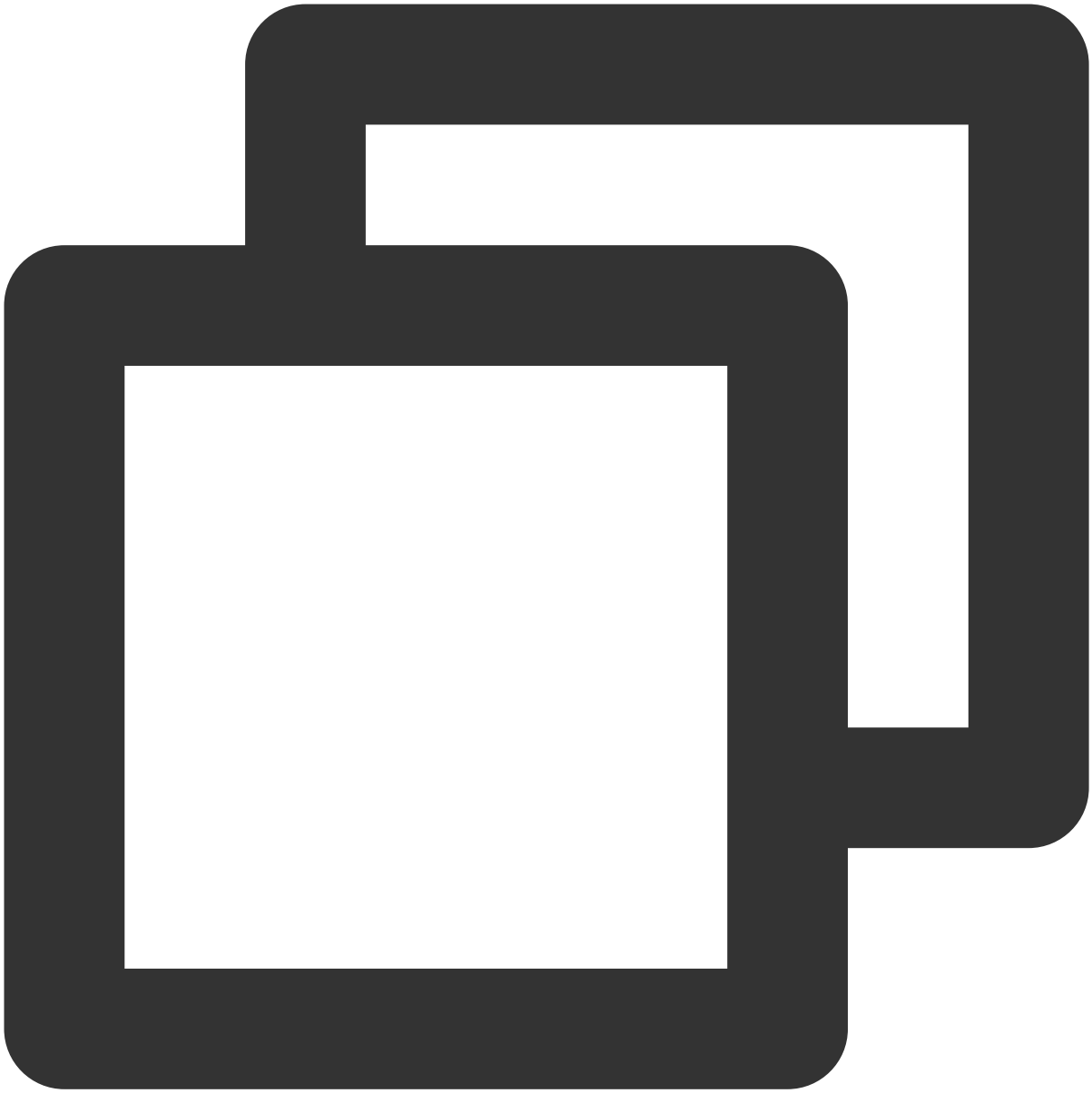


```
initWithRenderSize:assetsDict: (XMagic)
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDi
```

3. Tencent Effect SDKが各フレームのデータを処理し、対応する処理結果を返します。



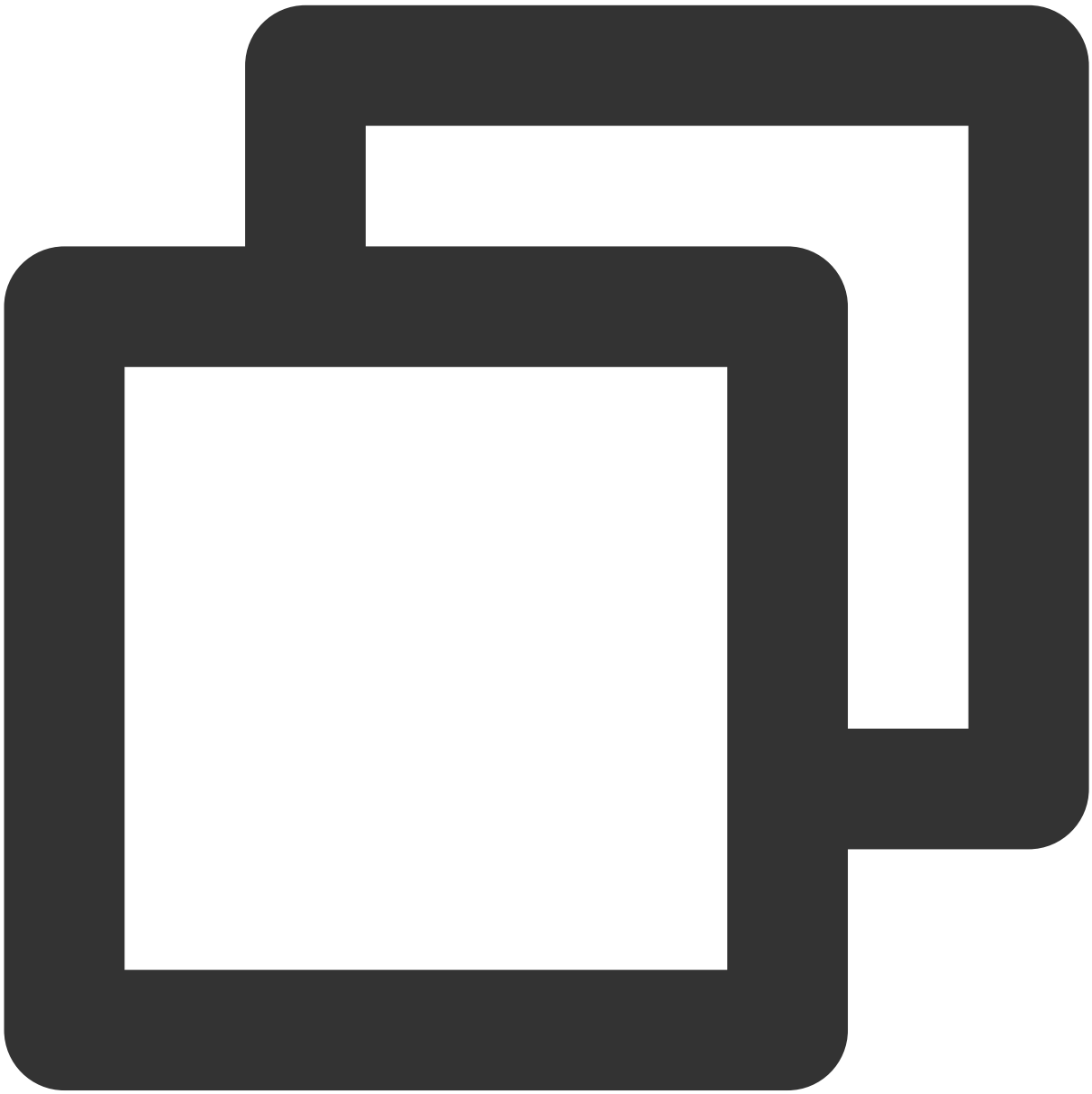
```
process: (XMagic)
```



```
// カメラコールバックでフレームデータを渡します
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:(CMSam

// オリジナルデータを取得し、各フレームのレンダリング情報を処理します
- (void)mycaptureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:(CMS

// CPUを使用してデータを処理します
- (YTProcessOutput*)processDataWithCpuFuc:(CMSampleBufferRef)inputSampleBuffer;

// GPUを使用してデータを処理します
- (YTProcessOutput*)processDataWithGpuFuc:(CMSampleBufferRef)inputSampleBuffer;
```

```
// Tencent Effect SDKでデータインターフェースを処理します
/// @param input処理データ情報を入力します
/// @return処理後のデータ情報を出力します
- (YTProcessOutput* _Nonnull)process:(YTProcessInput * _Nonnull)input;
```

4. Tencent Effect SDKをリリースします。



```
deinit (XMagic)
// SDKリソースをリリースする必要がある場所を呼び出します
[self.beautyKit deinit]
```

説明

上記の手順が完了すると、ユーザーは自身の実際のニーズに応じて表示のタイミングおよびその他のデバイスの関連環境を制御できるようになります。

よくあるご質問

質問1：コンパイルエラー「unexpected service error: build aborted due to an internal error: unable to write manifest to-xxxx-manifest.xcbuild': mkdir(/data, S_IRWXU | S_IRWXG | S_IRWXO): Read-only file system (30):」が発生しました。

1. **File** > **Project settings** > **Build System**と進み、**Legacy Build System**を選択します。
2. Xcode 13.0++の場合は**File** > **Workspace Settings**で**Do not show a diagnostic issue about build system deprecation**にチェックを入れます。

質問2：iOSでのリソースインポート実行後のエラー：「Xcode 12.Xバージョンのコンパイルで Building for iOS Simulator, but the linked and embedded framework '.framework'...と表示される」が発生しました。

Build Settings > **Build Options** > **Validate Workspace** をYesに変更し、再度**実行**をクリックします。

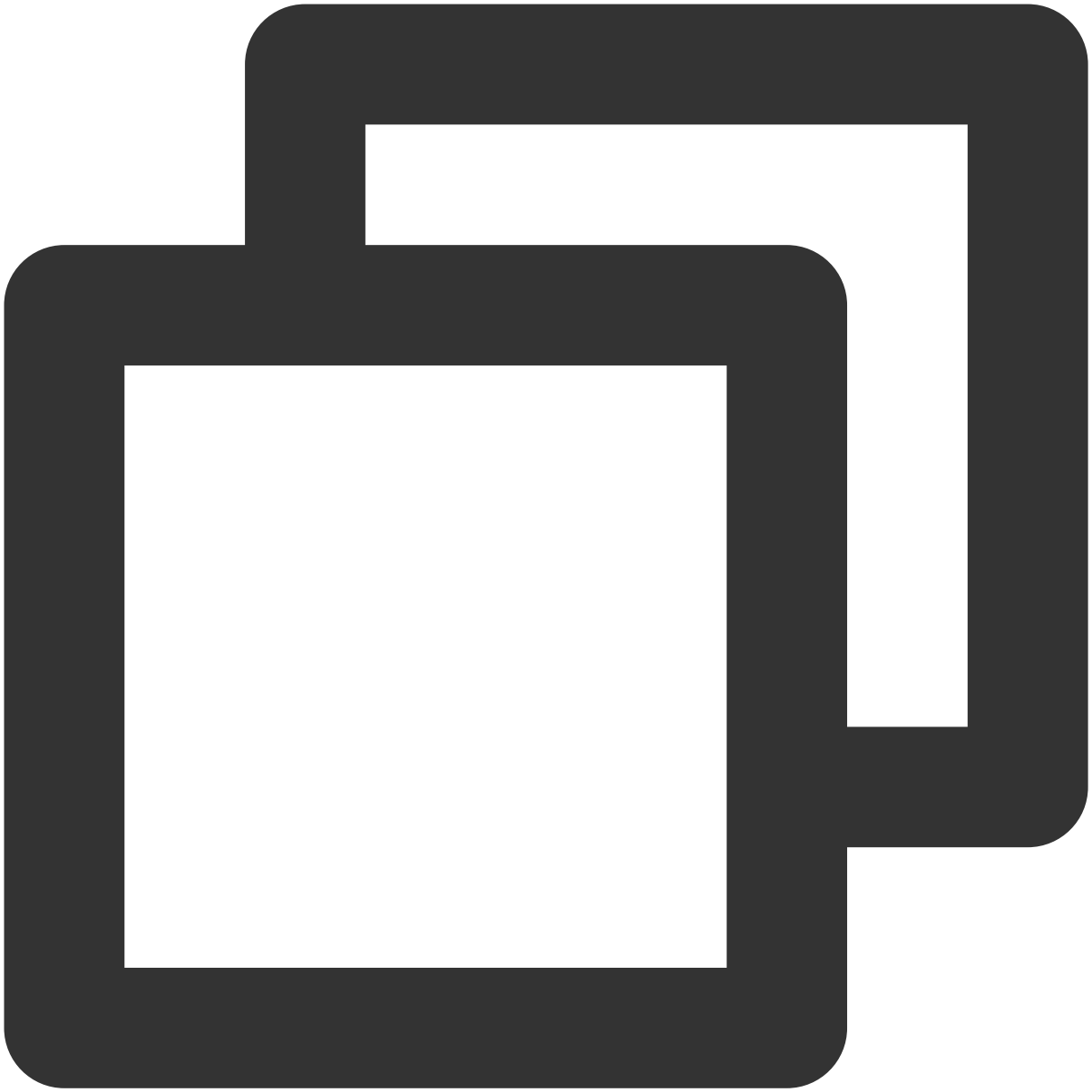
説明

Validate WorkspaceをYesに変更するとコンパイルが完了します。再びNoに変更しても正常に実行できます。そのため、ここではこの問題が発生した場合にのみ注意してください。

質問3：フィルター設定が反応しません。

設定値が正しいかどうか確認してください。範囲は0～100ですが、値が小さすぎると効果がわかりづらい場合があります。

質問4：iOS Demoのコンパイルで、dSYMを生成する際にエラーが発生します。



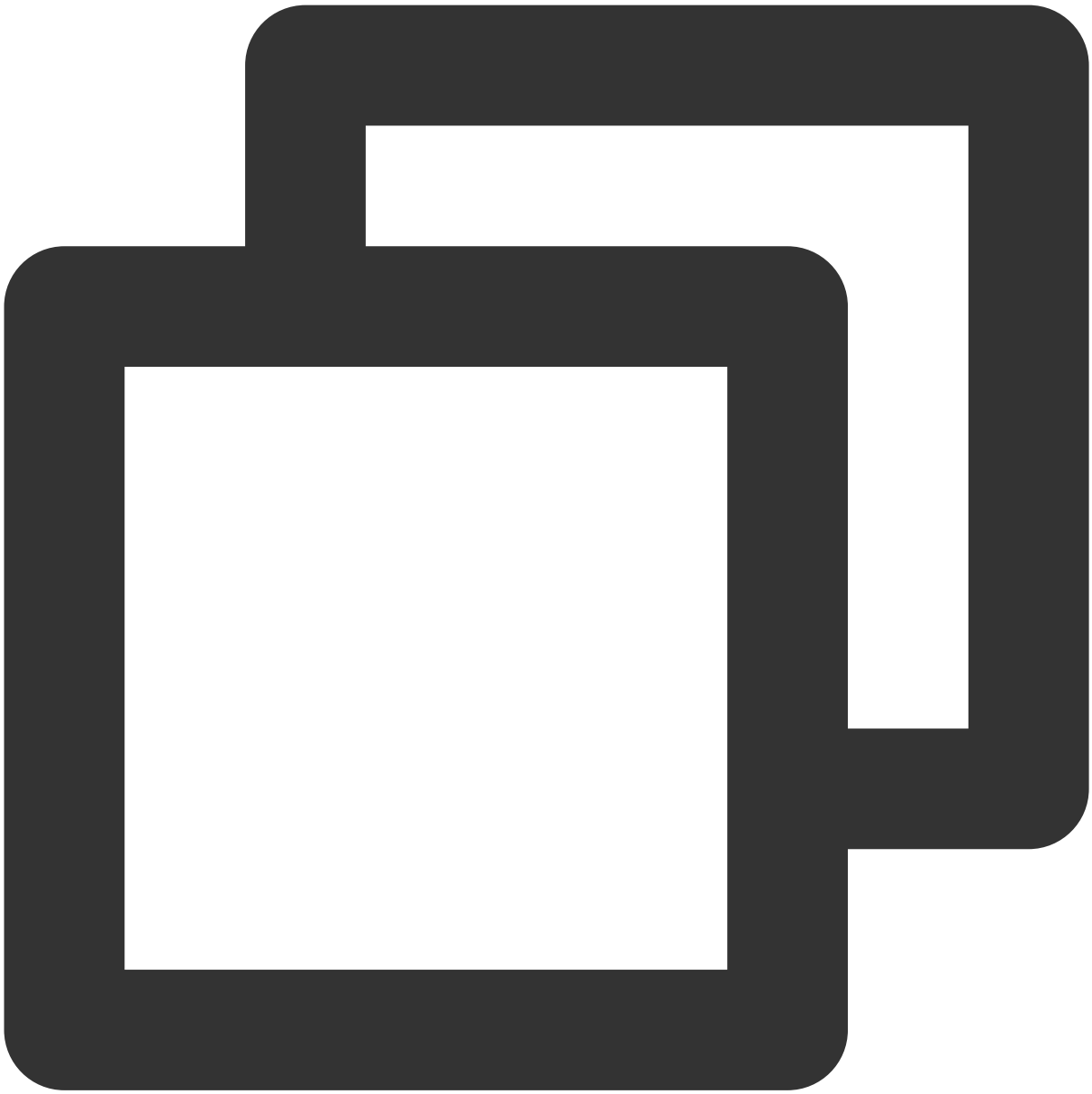
```
PhaseScriptExecution CMake\\ PostBuild\\ Rules build/XMagicDemo.build/Debug-iphoneo
    cd /Users/zhenli/Downloads/xmagic_s106
    /bin/sh -c /Users/zhenli/Downloads/xmagic_s106/build/XMagicDemo.build/Debug-iph

Command /bin/sh failed with exit code 1
```

問題の原因： `libpag.framework`と`Masonary.framework` の再署名に失敗したことが原因です。

解決方法：

1.1 `demo/copy_framework.sh` を開きます。

1.2 次のコマンドを使用してローカルマシンのcmakeのパスを確認し、 `$(which cmake)` をローカルcmakeの絶対パスに変更します。



```
which cmake
```

1.3 `Apple Development: ......` をすべてご自身の署名に置き換えます。

Android

最終更新日：：2022-12-19 16:42:43

統合の準備

1. SDKのダウンロードを実行し、解凍します。
2. **以下のファイルを準備**します。

ファイルタイプ	説明
xmagic-xxx.aar	SDK。入力必須
../assets/	アルゴリズムモデル、素材リソースパック。入力必須
../jniLibs	soライブラリ。入力必須

リソースのインポート

- 手動統合
- Mavenの統合
- 動的ダウンロードと統合

統合

- 上記で準備したすべての `.aar` ファイルをappプロジェクトの `libs` ディレクトリ下に追加します。
- SDKパッケージのassets/ディレクトリにあるすべてのリソースを `../src/main/assets` ディレクトリにコピーします。SDKパッケージのMotionResフォルダにリソースがある場合は、このフォルダを `../src/main/assets` ディレクトリにコピーします。
- jniLibsフォルダをプロジェクトの `../src/main/jniLibs` ディレクトリ下にコピーします。

インポート方法

appモジュールの `build.gradle` を開き、依存参照を追加します。

```
android{
...
defaultConfig {
applicationIdを「権限承認licにバインドしたパッケージ名に変更」します
....
}
```

```
packagingOptions {
pickFirst '**/libc++_shared.so'
}
}
dependencies{
...
compile fileTree(dir: 'libs', include: ['*.jar','*.aar'])//追加*.aar
}
```

注意
プロジェクトにGoogleのGsonライブラリを統合していない場合は、次の依存も追加する必要があります。

```
dependencies{
implementation 'com.google.code.gson:gson:2.8.2'
}
```

全体フロー

手順1：認証

1. 権限の承認を申請し、License URLとLicense KEYを取得します。

   注意：
   正常な状況では、Appのネットワーク接続が一度成功すれば認証フローは完了するため、Licenseファイルをプロジェクトのassetsディレクトリに保存する**必要はありません**。ただし、Appがネットワークに未接続の状態でSDKの関連機能を使用する必要がある場合は、Licenseファイルをダウンロードしてassetsディレクトリに保存し、最低保証プランとすることができます。この場合、Licenseファイル名は必ず `v_cube.license` としなければなりません。

2. 関連業務モジュールの初期化コードの中でURLとKEYを設定し、licenseのダウンロードをトリガーします。使用する直前になってダウンロードすることは避けてください。ApplicationのonCreateメソッドでダウンロードをトリガーすることもできますが、この時点でネットワークの権限がない可能性や、ネットワーク接続の失敗率が比較的高い可能性があるため、推奨しません。

```
//ダウンロードのトリガーまたはlicenseの更新のみが目的であり、認証結果には関心がない場合は、4つ目のパラメータにはnullを渡します。
TELicenseCheck.getInstance().setXMagicLicense(context, URL, KEY, null);
```

3. その後、実際に美顔機能を使用する前に(例えばDemoの `LaunchActivity.java` など)認証を行います。

```
// soライブラリがネットワークからダウンロードしたものの場合は、TELicenseCheck.getInstance().setTELicenseを呼び出す前にsoのパスを設定しておかなければ、認証に失敗する場合があります。
// XmagicApi.setLibPathAndLoad(validLibsDirectory);
// soがapkパッケージ内にある場合は、上記のメソッドを呼び出す必要はありません。
TELicenseCheck.getInstance().setTELicense(context, URL, KEY, new TELicenseCheckListener() {
@Override
public void onLicenseCheckFinish(int errorCode, String msg) {
//注意：このコールバックは呼び出しスレッド内にあるとは限りません
if (errorCode == TELicenseCheck.ERROR_OK) {
//認証に成功しました
}else{
//認証に失敗しました
}
}
});
```

認証errorCodeの説明：

エラーコード	説明
0	成功です。Success
-1	入力パラメータが無効です（例：URLまたはKEYが空など）
-3	ダウンロードの段階で失敗しました。ネットワークの設定を確認してください
-4	ローカルから読み取ったTE権限承認情報が空です。IOの失敗による可能性があります
-5	読み取ったVCUBE TEMP Licenseファイルの内容が空です。IOの失敗による可能性があります
-6	`v_cube.license` ファイルのJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-7	署名の検証に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください

エラーコード	説明
-8	復号に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-9	TELicenseフィールド内のJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-10	ネットワークから解析したTE権限承認情報が空です。Tencent Cloudチームに連絡して処理を依頼してください
-11	TE権限承認情報をローカルファイルに書き込む際に失敗しました。IOの失敗による可能性があります
-12	ダウンロードに失敗しました。ローカルassetの解析も失敗しました
-13	認証に失敗しました。soがパッケージ内にあるか、またはsoパスが正しく設定されているかを確認してください
3004/3005	権限承認が無効です。Tencent Cloudチームに連絡して処理を依頼してください
3015	Bundle Id / Package Name が一致しません。Appで使用しているBundle Id / Package Nameが申請時のものと同じか、正しい権限承認ファイルを使用しているかを確認してください
3018	権限承認ファイルが期限切れです。Tencent Cloudに更新を申請する必要があります
その他	Tencent Cloudチームに連絡して処理を依頼してください

手順2：リソースのコピー

1. リソースファイルがassetsディレクトリ内にある場合は、使用する前にappのプライベートディレクトリにコピーしておく必要があります。事前にコピーしておくか、または前の手順の認証成功のコールバックの中でコピー操作を行うこともできます。サンプルコードは、Demoの `LaunchActivity.java` です。

```
XmagicResParser.setResPath(new File(getFilesDir(), "xmagic").getAbsolutePath()
);
//loading
//リソースファイルのプライベートディレクトリへのコピーは1回のみ行います
XmagicResParser.copyRes(getApplicationContext());
```

2. リソースファイルがネットワークから動的ダウンロードしたものの場合は、ダウンロード成功後にリソースファイルパスを設定する必要があります。サンプルコードは、Demoの `LaunchActivity.java` です。

```
XmagicResParser.setResPath(ダウンロードしたリソースファイルのローカルパス);
```

手順3：SDKの初期化および使用方法

Tencent Effect SDK使用のライフサイクルはおおむね次のとおりです。

1. 美顔UIデータを作成します。Demoプロジェクトのものを参照できます。
`XmagicResParser.java,XmagicUIProperty.java,XmagicPanelDataManager.java` コード。

2. プレビューレイアウトにGLSurfaceViewを追加します。

```
<android.opengl.GLSurfaceView
android:id="@+id/camera_gl_surface_view"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

3. （オプション）カメラをクイック実装します。
Demoプロジェクトのcom.tencent.demo.cameraディレクトリをプロジェクトにコピーします。 `PreviewMgr` クラスを利用してカメラ機能をクイック実装します。実装の詳細についてはDemoプロジェクトの `MainActivity.java` をご参照ください。

```
//カメラ初期化
mPreviewMgr = new PreviewMgr();
//レイアウトしたGlSurfaceViewサンプルをカメラツールクラスに渡します
mPreviewMgr.onCreate(mGlSurfaceView,false);
//プレビューテクスチャデータのコールバック関数を登録します
mPreviewMgr.setCustomTextureProcessor((textureId, textureWidth, textureHeight)
-> {
if (mXmagicApi == null) {
return textureId;
}
//美顔sdkを呼び出してレンダリングします
int outTexture = mXmagicApi.process(textureId, textureWidth, textureHeight);
return outTexture;
});
//Activityの中のonResumeメソッドでカメラを有効にします
mPreviewMgr.onResume(this, 1280, 720);
```

4. 美顔SDKを初期化します。Activityの `onResume()` メソッドに保存することをお勧めします。

```
mXmagicApi = new XmagicApi(this, XmagicResParser.getResPath(),new XmagicApi.OnX
magicPropertyErrorListener());
```

- **パラメータ**

パラメータ	意味
Context context	コンテキスト
String resDir	リソースファイルディレクトリ。詳細については、手順2をご参照ください
OnXmagicPropertyErrorListener errorListener	コールバック関数実装クラス

- **戻り値**
  エラーコードの意味についてはAPIドキュメントをご参照ください。

5. 素材プロンプトのコールバック関数を追加します（メソッドのコールバックはサブスレッドで実行される場合があります）。一部の素材はユーザーに、うなずく、手を伸ばす、指ハートなどを促します。このコールバックは類似のプロンプトの表示に用いられます。

```
mXmagicApi.setTipsListener(new XmagicTipsListener() {
final XmagicToast mToast = new XmagicToast();
@Override
public void tipsNeedShow(String tips, String tipsIcon, int type, int duration)
{
mToast.show(MainActivity.this, tips, duration);
}
@Override
public void tipsNeedHide(String tips, String tipsIcon, int type) {
mToast.dismiss();
}
});
```

6. 美顔SDKが各フレームのデータを処理し、対応する処理結果を返します。

```
int outTexture = mXmagicApi.process(textureId, textureWidth, textureHeight);
```

7. 指定するタイプの美顔エフェクトの数値を更新します。

```
// 使用可能な入力パラメータのプロパティはXmagicResParser.parseRes()から取得できます
mXmagicApi.updateProperty(XmagicProperty<?> p);
```

8. 美顔SDKをPauseします。Activityの `onPause()` ライフサイクルにバインドすることをお勧めします。

```
//ActivityがonPauseの場合に呼び出します。OpenGLスレッドで呼び出す必要があります
mXmagicApi.onPause();
```

9. 美顔SDKをリリースします。Activityの `onDestroy()` ライフサイクルにバインドすることをお勧めします。

```
//このメソッドはGLスレッドで呼び出す必要があることにご注意ください
mXmagicApi.onDestroy()
```

Tencent Effectをライブブロードキャスト SDKに統合
ios

最終更新日：：2023-02-27 14:18:15

統合の準備

1. Demoパッケージをダウンロードして解凍します。
2. Demoプロジェクトのxmagicモジュール（bundle、XmagicIconRes、Xmagicフォルダ）を実際のプロジェクトにインポートします。
3. **使用しているXMagic SDKのバージョンが2.5.0より古い場合は、**SDKディレクトリ内の `libpag.framework` 、 `Masonry.framework` 、 `XMagic.framework` 、 `YTCommonXMagic.framework` をインポートします。**使用しているXMagic SDKのバージョンが2.5.1およびそれ以降の場合は、**SDKディレクトリ内の `libpag.framework` 、 `Masonry.framework` 、 `XMagic.framework` 、 `YTCommonXMagic.framework` 、 `Audio2Exp.framework` 、 `TEFFmpeg.framework` をインポートします。
4. frameworkの署名は、**General--> Masonry.framework**と**libpag.framework**は**Embed & Sign**を選択します。**YTCommonXMagic.framework** は、バージョン2.5.1より古い場合は**Do Not Embed**を選択し、バージョン2.5.1およびそれ以降の場合は**Embed & Sign**を選択します。
5. Bundle IDを、発行された権限と同じものに変更します。

開発者環境要件

開発ツールXCode 11およびそれ以上：App Storeまたはダウンロードアドレスをクリックします。
推奨実行環境：
デバイス要件：iPhone 5およびそれ以上である必要があります。iPhone 6およびそれ未満の場合は、フロントカメラのサポートを最大720pとし、1080pはサポートしていません。
システム要件：iOS 10.0およびそれ以降のバージョン。

C/C++レイヤー開発環境

XCodeはデフォルトではC++環境となります。

タイプ	依存ライブラリ
システム依存ライブラリ	Accelerate AssetsLibrary AVFoundation

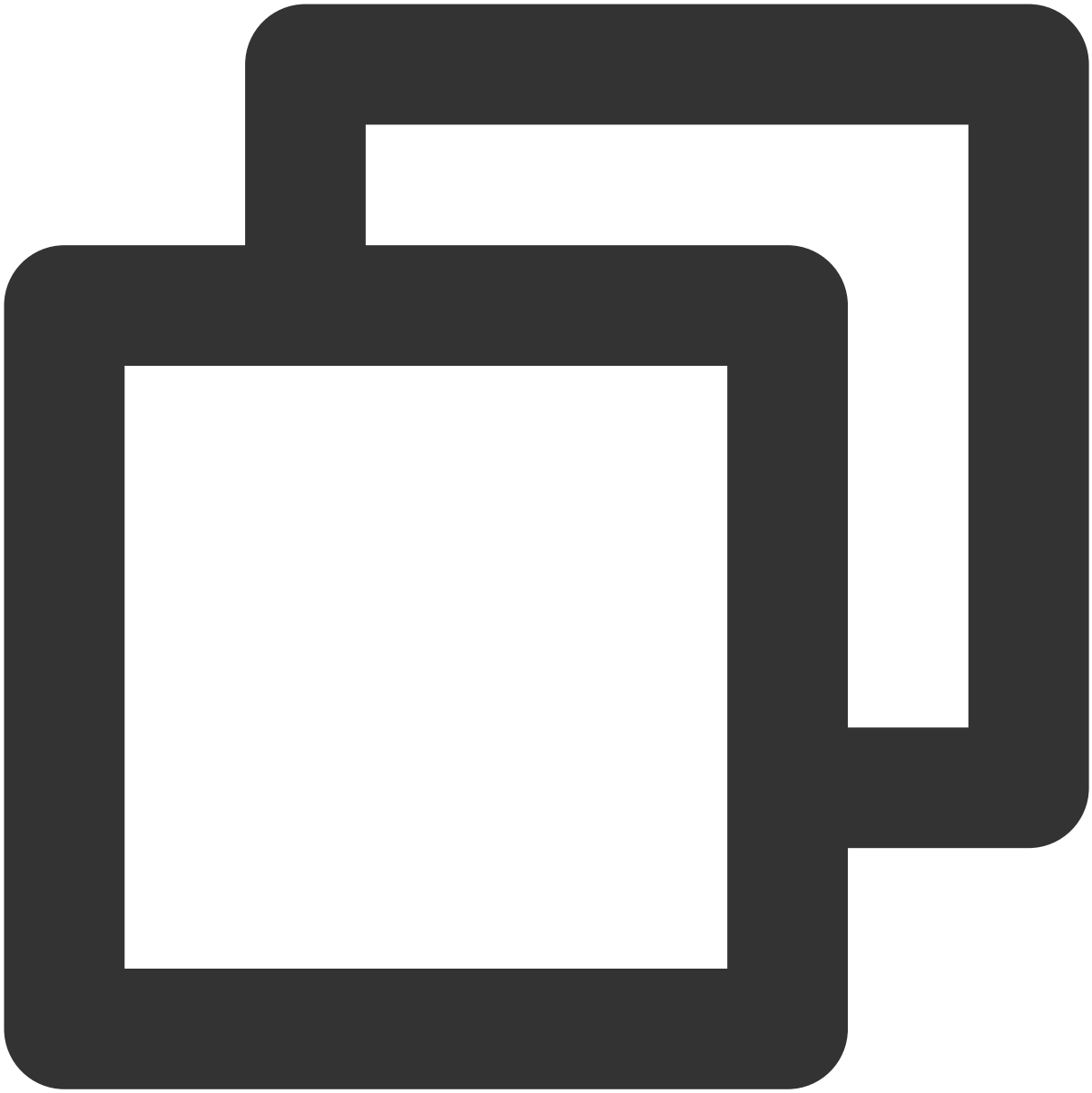
	CoreMedia CoreFoundation CoreML Foundation JavaScriptCore libc++.tbd libz.b libresolv.tbd libsqlite3.0.tbd MetalPerformanceShaders MetalKit MobileCoreServices OpneAL OpneGLES ReplayKit SystemConfiguration UIKit
付属ライブラリ	YTCommon（認証静的ライブラリ） XMagic（美顔静的ライブラリ） libpag（ビデオデコード動的ライブラリ） Masonry（ウィジェットレイアウトライブラリ） TXLiteAVSDK_Professional TXFFmpeg TXSoundTouch Audio2Exp（xmagic sdk versionは2.5.1およびそれ以降のバージョンのみ） TEFFmpeg（xmagic sdk versionは2.5.1およびそれ以降のバージョンのみ）

SDKインターフェースの統合

ステップ1およびステップ2については、Demoプロジェクトの `ThirdBeautyViewController` クラスの `viewDidLoad` 、 `buildBeautySDK` メソッドを参照できます。 `AppDelegate` クラスの `application` メソッドはXmagic認証を実行します。
ステップ4からステップ7までは、Demoプロジェクトの `ThirdBeautyViewController` 、 `BeautyView` クラスの関連インスタンスコードを参照できます。

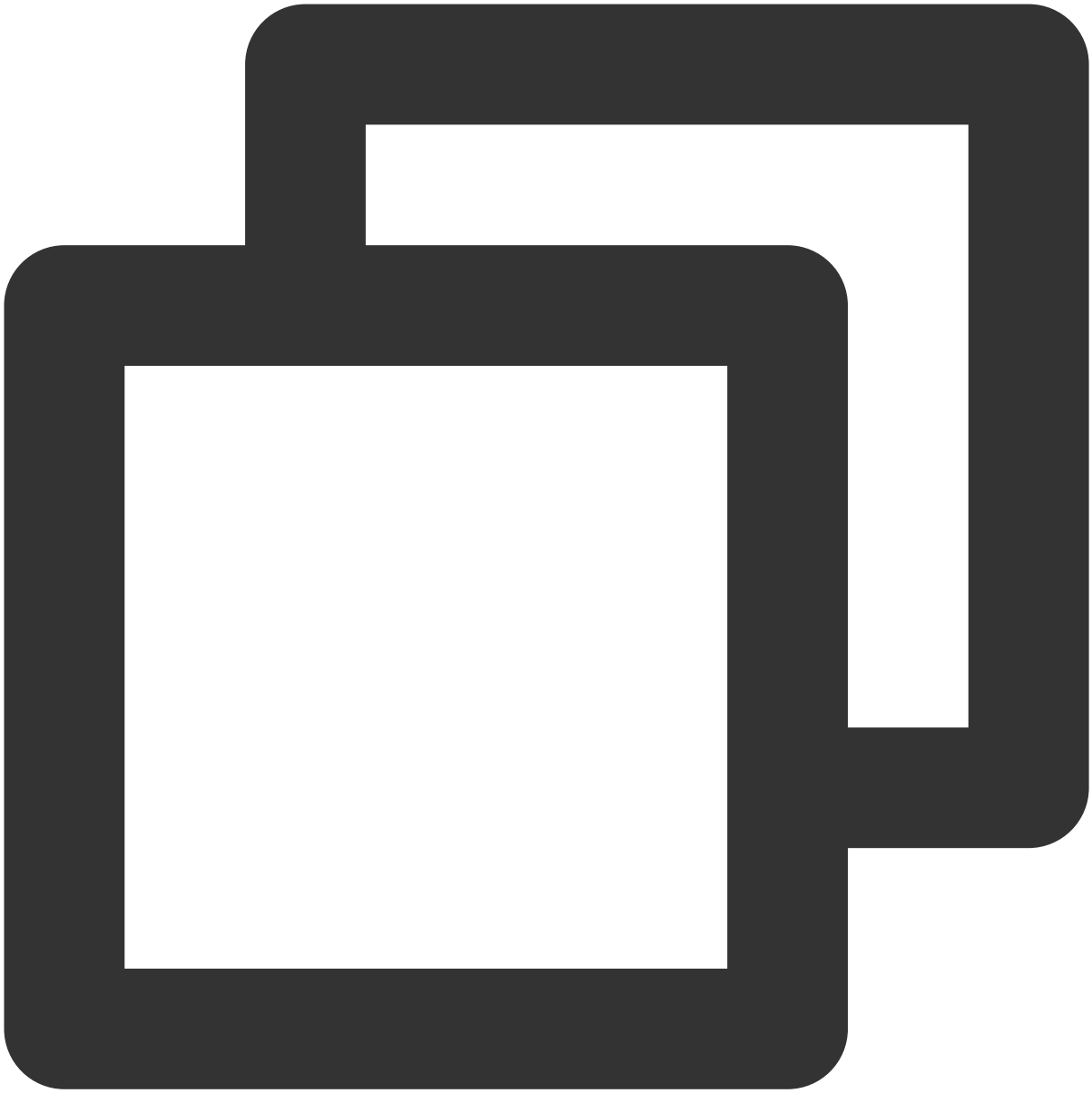
ステップ1：権限の初期化

1. 初めにプロジェクトの `AppDelegate` の `didFinishLaunchingWithOptions` に次の認証コードを追加します。このうち `LicenseURL` 、 `LicenseKey` はTencent Cloud公式サイトに権限承認を申請した際の情報とします。



```
[TXLiveBase setLicenceURL:LicenseURL key:LicenseKey];
```

2. Xmagicの認証：関連業務モジュールの初期化コードの中でURLとKEYを設定し、licenseのダウンロードをトリガーします。使用する直前になってダウンロードすることは避けてください。あるいは `AppDelegate` の `didFinishLaunchingWithOptions` メソッドでダウンロードをトリガーすることもできます。このうち `LicenseURL` と `LicenseKey` はコンソールでLicenseをバインドした際に生成された権限承認情報です。SDKのバージョンが2.5.1より古い場合、 `TELicenseCheck.h` は `XMagic.framework` 内にあります。SDKのバージョンが2.5.1およびそれ以降の場合、 `TELicenseCheck.h` は `YTCommonXMagic.framework` 内にあります。



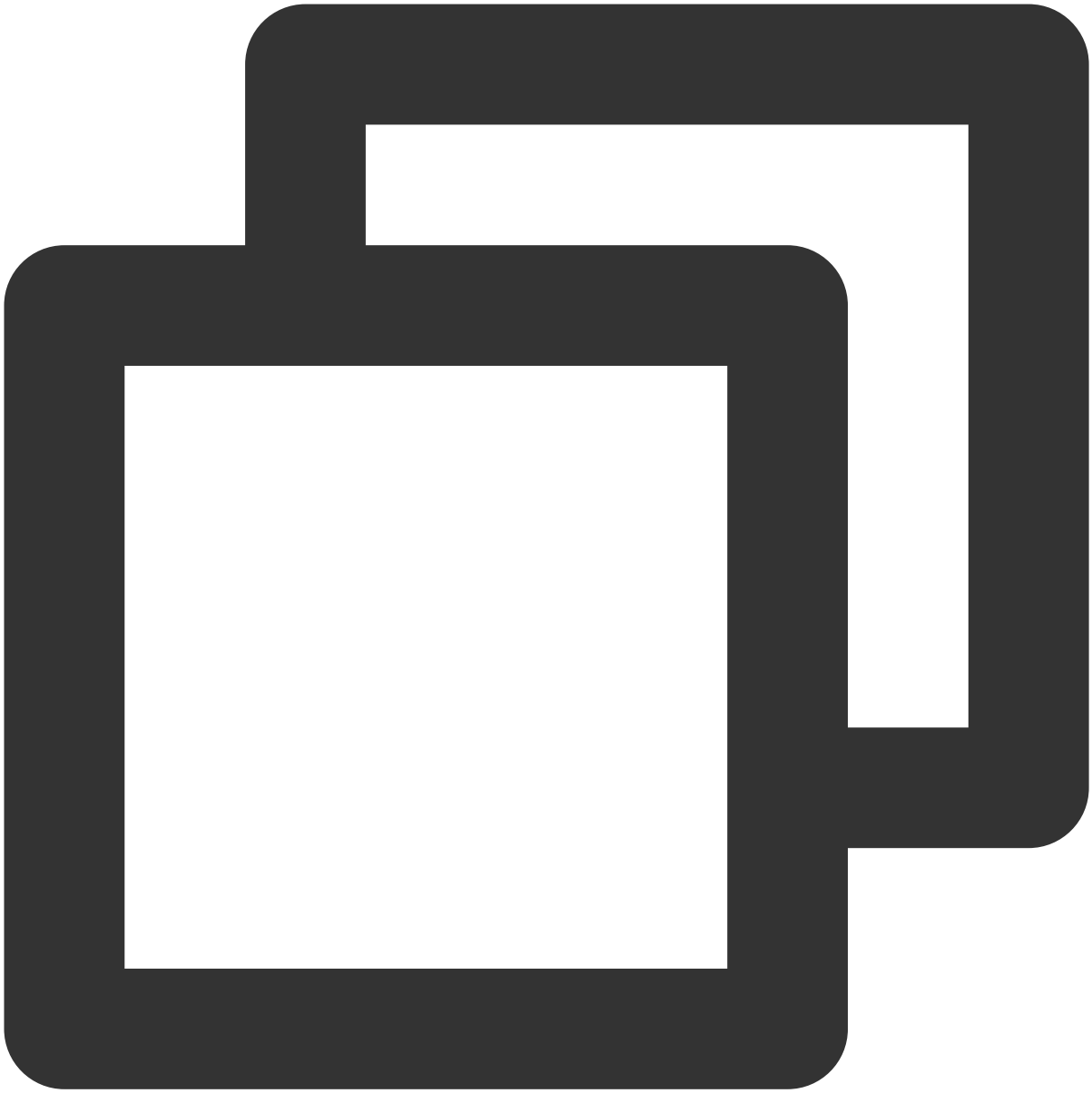
```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authr
    if (authresult == TELicenseCheckOk) {
        NSLog(@"認証成功");
    } else {
        NSLog(@"認証失敗");
    }
}];
```

認証errorCode説明：

エラーコー	説明

ド	
0	成功です。Success
-1	入力パラメータが無効です（例：URLまたはKEYが空など）
-3	ダウンロードの段階で失敗しました。ネットワークの設定を確認してください
-4	ローカルから読み取ったTE権限承認情報が空です。IOの失敗による可能性があります
-5	読み取ったVCUBE TEMP Licenseファイルの内容が空です。IOの失敗による可能性があります
-6	v_cube.licenseファイルのJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-7	署名の検証に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-8	復号に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-9	TELicenseフィールド内のJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-10	ネットワークから解析したTE権限承認情報が空です。Tencent Cloudチームに連絡して処理を依頼してください
-11	TE権限承認情報をローカルファイルに書き込む際に失敗しました。IOの失敗による可能性があります
-12	ダウンロードに失敗しました。ローカルassetの解析も失敗しました
-13	認証に失敗しました
その他	Tencent Cloudチームに連絡して処理を依頼してください

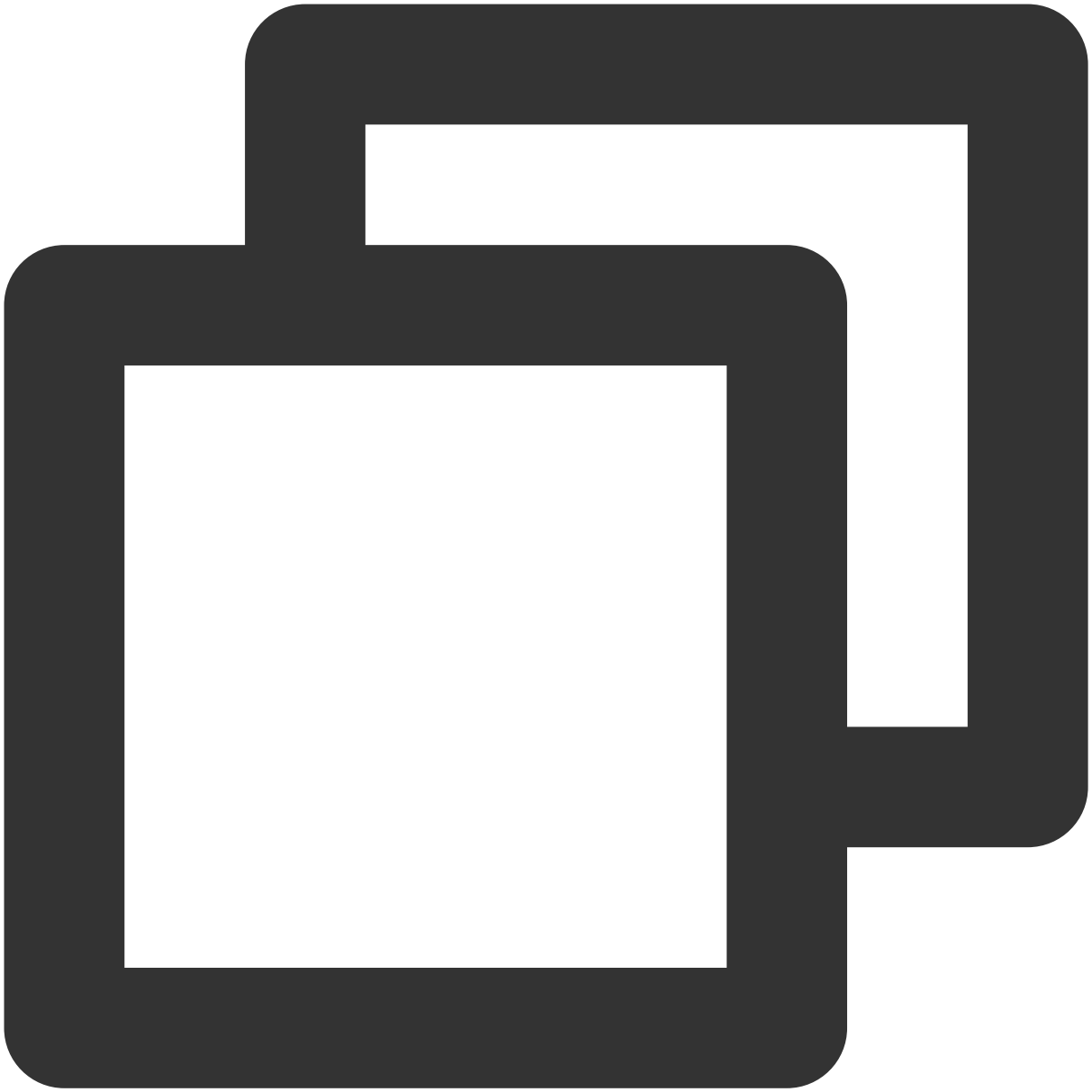
ステップ2：SDK素材リソースパスの設定



```
CGSize previewSize = [self getPreviewSizeByResolution:self.currentPreviewResolution
NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirecto
beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_config
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !isD
    NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfig
    NSError *jsonError;
    NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8StringEncodin
    beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
```

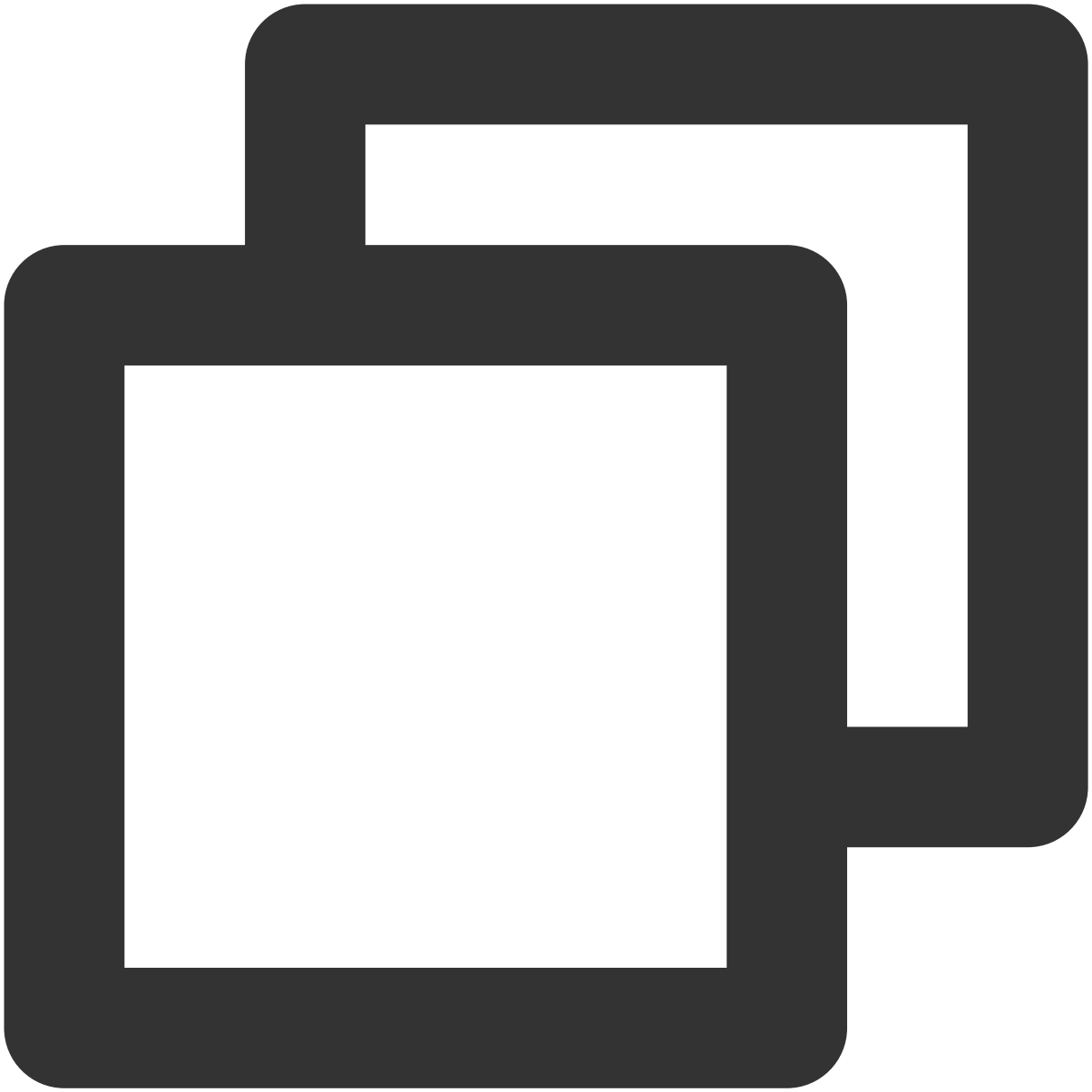
```
                                                options:NSJSONReadingMutableContainers
                                                error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle] bundlePath],
                             @"tnn_"
                             @"beauty_config":beautyConfigJson
};
// Init beauty kit
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDi
```

ステップ3：ログおよびイベント監視の追加



```
    // Register log
    [self.beautyKit registerSDKEventListener:self];
    [self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL
```

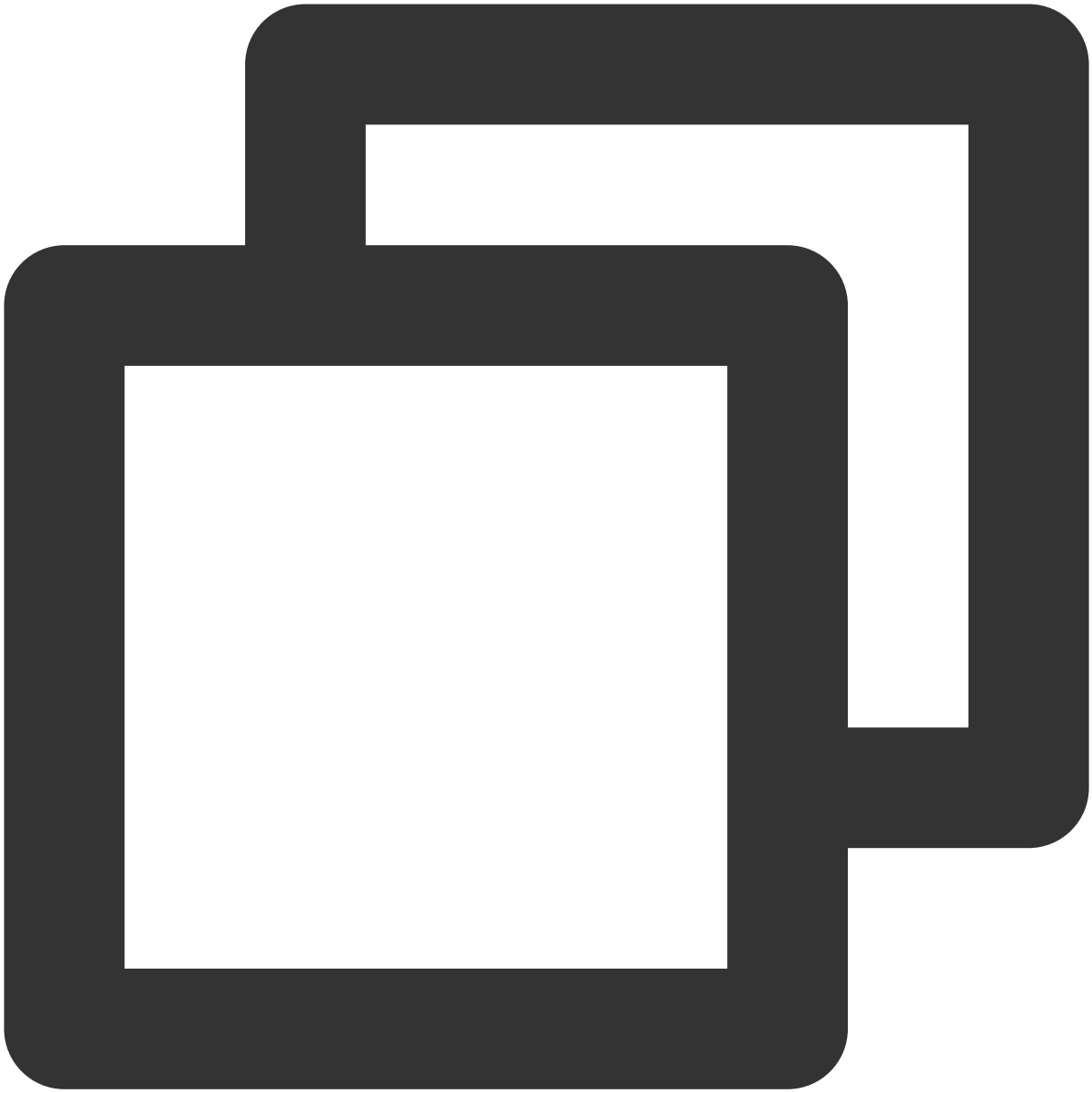
ステップ4：美顔の各種効果の設定



```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString *
```

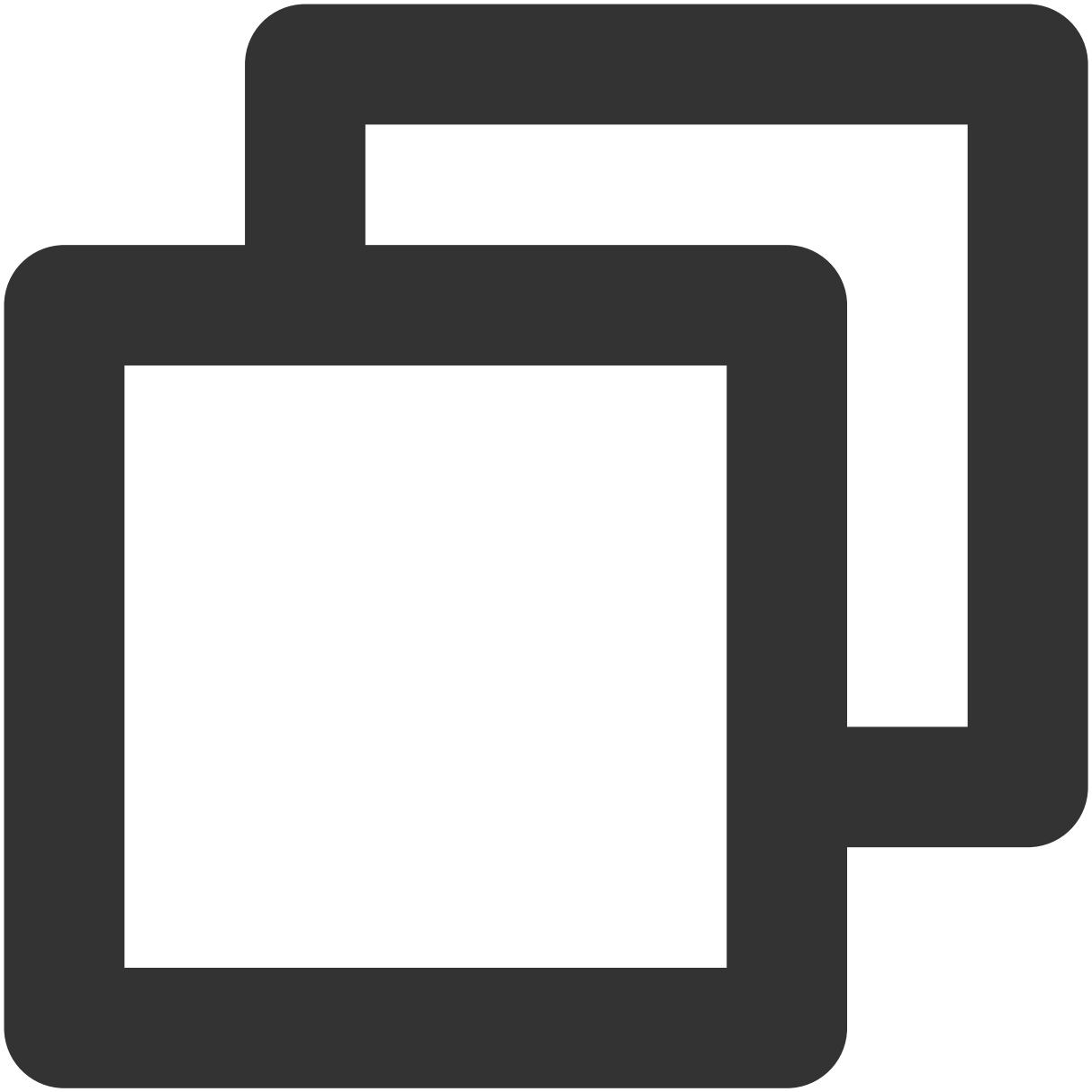
ステップ5：レンダリング処理の実施

ビデオフレームコールバックインターフェースで、YTProcessInputを作成してSDKに渡し、レンダリング処理を行います。DemoのThirdBeautyViewControllerを参照できます。



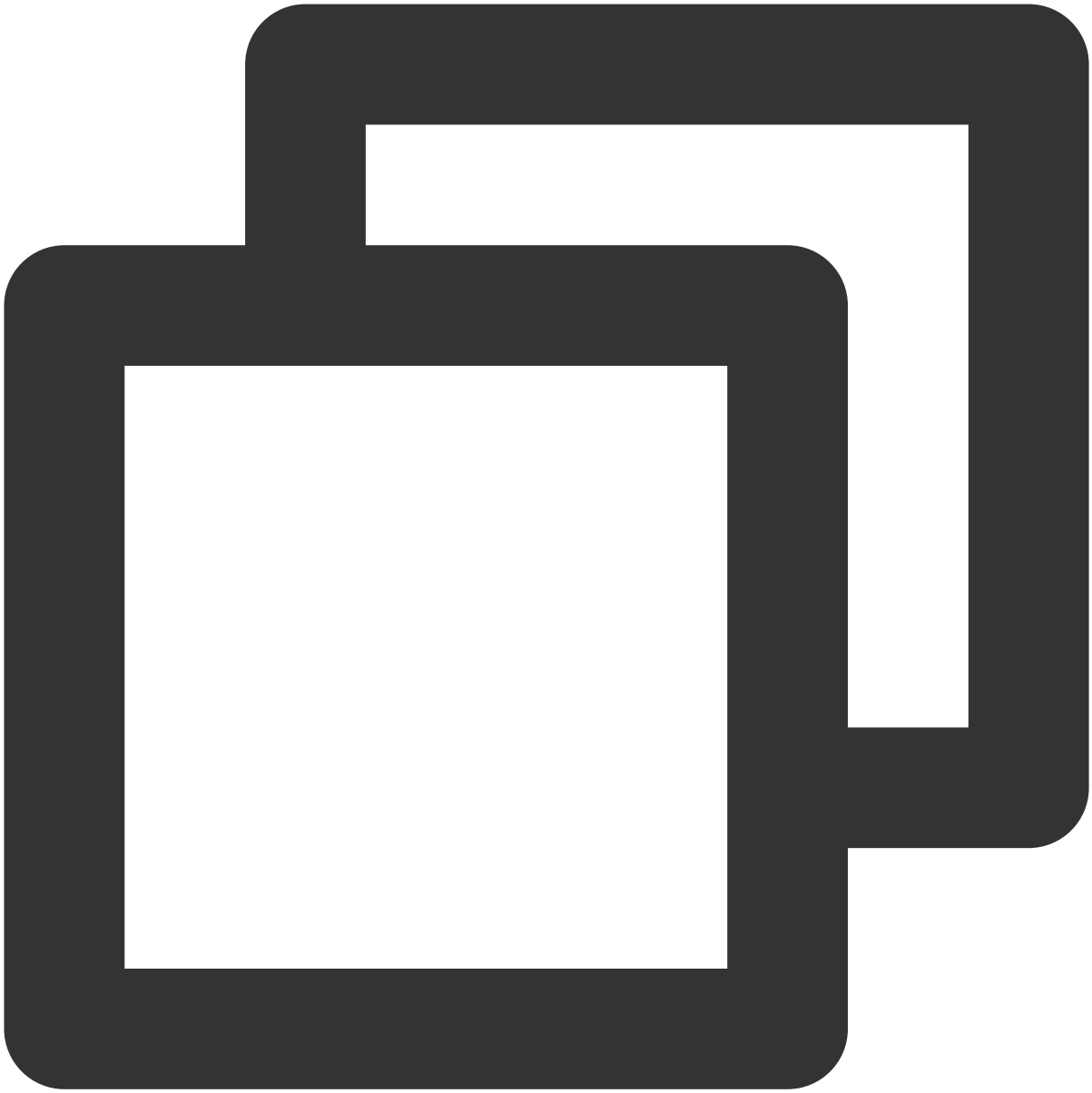
```
[self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft withOrientati
```

ステップ6：SDKの一時停止/再開



```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

ステップ7：レイアウトにSDK美顔パネルを追加



```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
if (@available(iOS 11.0, *)) {
    gSafeInset = [UIApplication sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
    {
        gSafeInset = UIEdgeInsetsZero;
    }
```

```
dispatch_async(dispatch_get_main_queue(), ^{
    //美顔オプションインターフェース
    _vBeauty = [[BeautyView alloc] init];
    [self.view addSubview:_vBeauty];
    [_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
        make.width.mas_equalTo(self.view);
        make.centerX.mas_equalTo(self.view);
        make.height.mas_equalTo(254);
        if(gSafeInset.bottom > 0.0){  // 全画面適用
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
        } else {
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
        }
    }];
    _vBeauty.hidden = YES;
});
```

Android

最終更新日：：2022-07-18 10:06:17

手順1：Demoプロジェクトの解凍

1. Tencent Effect TEを統合したMLVB Demoプロジェクトをダウンロードします。このDemoは、Tencent Effect SDK S1-04パッケージに基づいて作成されています。
2. リソースを置き換えます。このDemoプロジェクトで使用されるSDKパッケージは実際のパッケージと同じではない可能性があるため、このDemoの関連SDKファイルを実際に使用するパッケージのSDKファイルに置き換えてください。具体的な操作は以下のとおりです：
   - xmagicモジュールのlibsディレクトリにある `.aar` ファイルを削除し、SDKのlibsディレクトリにある.aar`ファイルをxmagicモジュールのlibsディレクトリにコピーします。
   - xmagicモジュールのassetsディレクトリにあるすべてのファイルを削除し、SDKの `assets/` ディレクトリにあるすべてのリソースをxmagicモジュールの `../src/main/assets` ディレクトリにコピーします。SDKパッケージのMotionResフォルダにリソースがある場合は、このフォルダを `../src/main/assets` ディレクトリにコピーします。
   - xmagicモジュールのjniLibsディレクトリにあるすべての.soファイルを削除し、SDKパッケージのjniLibsで対応する.soファイルを見つけて（SDKのjinLibsフォルダにあるarm64-v8aおよびarmeabi-v7aの `.so` ファイルが圧縮パッケージに存在しているため、先に解凍してください）、xmagicモジュールの `../src/main/jniLibs` ディレクトリにコピーします。
3. Demoプロジェクトのxmagicモジュールを実際のプロジェクトにインポートします。

手順2：appモジュールのbuild.gradleを開く

1. applicationIdを、テスト用に申請した権限と同じパッケージ名に変更します。
2. gson依存設定を追加します。

```
configurations {
all*.exclude group: 'com.google.code.gson'
}
```

手順3：SDKインターフェースの統合

DemoプロジェクトのThirdBeautyActivityクラスを参照できます。

1. 権限承認：

```
//認証に関する注意事項とエラーコードの詳細については、https://www.tencentcloud.com/document/product/1143/45385#step-1.-authenticateをご参照ください。
XMagicImpl.checkAuth((errorCode, msg) -> {
if (errorCode == TELicenseCheck.ERROR_OK) {
showLoadResourceView();
}else{
TXCLog.e(TAG, "認証に失敗しました。認証urlおよびkeyをご確認ください" + errorCode + " "
+ msg);
}
});
```

2. 素材の初期化：

```
private void showLoadResourceView() {
if (XmagicLoadAssetsView.isCopyedRes) {
XmagicResParser.parseRes(getApplicationContext());
initXMagic();
}else{
XmagicLoadAssetsView loadAssetsView = new XmagicLoadAssetsView(this);
loadAssetsView.setOnAssetsLoadFinishListener(() -> {
XmagicResParser.parseRes(getApplicationContext());
initXMagic();
});
}
}
```

3. プッシュ設定の有効化：

```
String userId = String.valueOf(new Random().nextInt(10000));
String pushUrl = AddressUtils.generatePushUrl(streamId, userId, 0);
mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.TXLiveMode_
RTC);
mLivePusher.enableCustomVideoProcess(true, V2TXLivePixelFormatTexture2D, V2TXLi
veBufferTypeTexture);
mLivePusher.setObserver(new V2TXLivePusherObserver() {
@Override
public void onGLContextCreated() {
}
@Override
public int onProcessVideoFrame(V2TXLiveDef.V2TXLiveVideoFrame srcFrame, V2TXLiv
eDef.V2TXLiveVideoFrame dstFrame) {
```

```
if (mXMagic != null) {
dstFrame.texture.textureId = mXMagic.process(srcFrame.texture.textureId, srcFra
me.width, srcFrame.height);
}
return srcFrame.texture.textureId;
}
@Override
public void onGLContextDestroyed() {
if (mXMagic != null) {
mXMagic.onDestroy();
}
}
});
mLivePusher.setRenderView(mPushRenderView);
mLivePusher.startCamera(true);
int ret = mLivePusher.startPush(pushUrl);
mLivePusher.startMicrophone();
```

4. **textureIdをSDKに渡し、レンダリング処理を実施**：

V2TXLivePusherObserverインターフェース の `onProcessVideoFrame(V2TXLiveDef.V2TXLiveVideoFrame srcFrame, V2TXLiveDef.V2TXLiveVideoFrame dstFrame)` メソッド内に次のコードを追加します。

```
if (mXMagic != null) {
dstFrame.texture.textureId = mXMagic.process(srcFrame.texture.textureId, srcFra
me.width,srcFrame.height);
}
return srcFrame.texture.textureId;
```

5. **SDKの一時停止/破棄**：

`onPause()` は美顔効果の一時停止に使用し、Activity/Fragmentライフサイクルメソッドにおいて実行できます。onDestroyメソッドはGLスレッドで呼び出す必要があります（onTextureDestroyedメソッドでXMagicImplオブジェクトの `onDestroy()` を呼び出すことができます）。その他の使用についてはDemoをご参照ください。

```
mXMagic.onPause(); //一時停止。ActivityのonPauseメソッドにバインドします
mXMagic.onDestroy(); // //破棄。GLスレッドで呼び出してください
```

6. **レイアウトにSDK美顔パネルを追加**：

```
<RelativeLayout
android:id="@+id/livepusher_bp_beauty_pannel"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_above="@+id/ll_edit_info" />
```

7. **パネルの初期化**：

```
private void initXMagic() {
if (mXMagic == null) {
mXMagic = new XMagicImpl(this, mBeautyPanelView);
}else{
mXMagic.onResume();
}
}
```

具体的な操作についてはDemoプロジェクトの `ThirdBeautyActivity.initXMagic();` メソッドをご参照ください。

Flutter

最終更新日：：2023-04-27 16:14:09

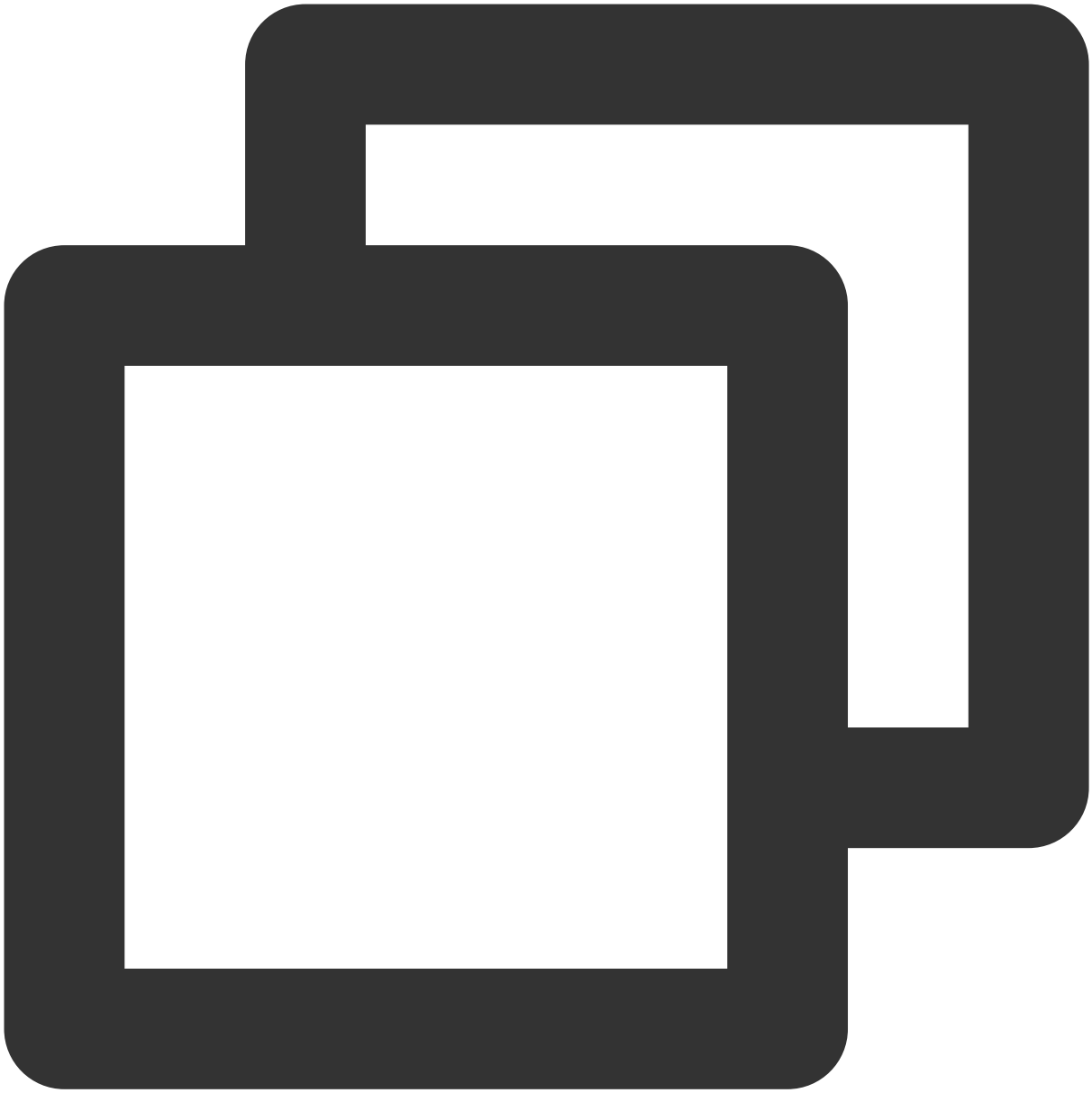
ステップ1：美顔リソースのダウンロードと統合

購入したパッケージに応じて、SDKのダウンロードを行い、ファイルをご自身のプロジェクトに追加します。

Android

iOS

1. appモジュール下でbuild.gradleファイルを見つけ、対応するパッケージのmaven参照アドレスを追加します。例えばS1-04パッケージを選択した場合は下記を追加します。

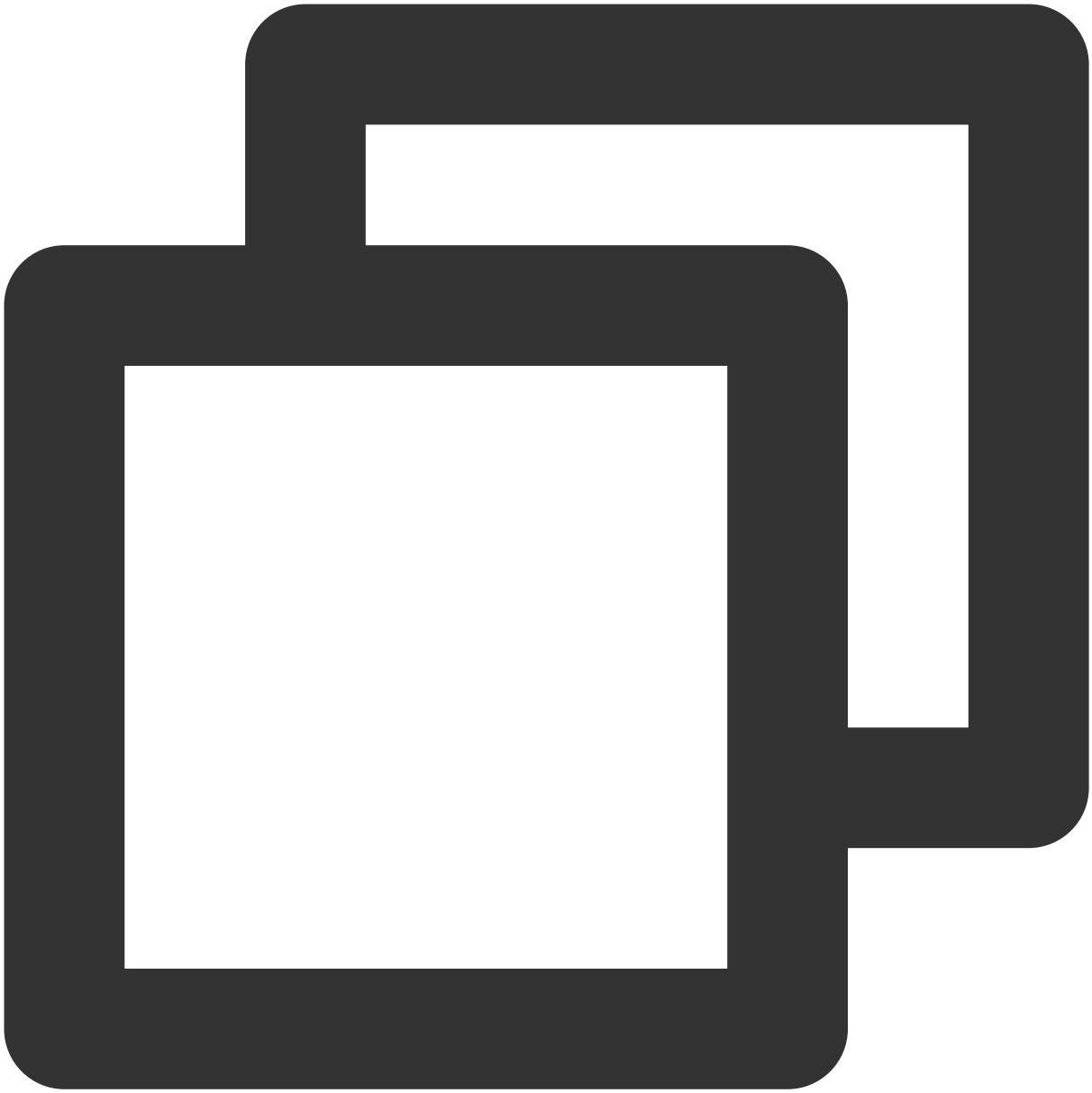


```
dependencies{
        implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
  }
```

各パッケージに対応するmavenアドレスについては、ドキュメントをご参照ください。

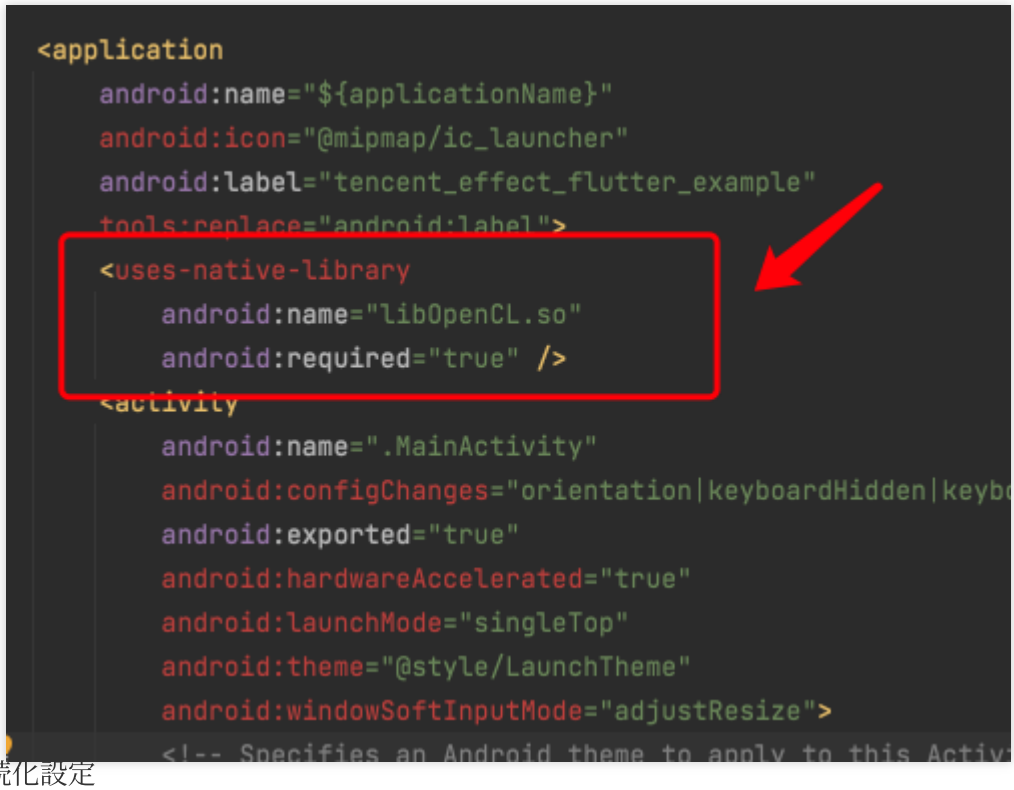
2. appモジュール下でsrc/main/assetsフォルダを見つけます。存在しない場合は作成し、ダウンロードしたSDKパッケージにMotionResフォルダがあるかどうかをチェックし、もしあればそのフォルダを `../src/main/assets` ディレクトリ下にコピーします。

3. appモジュール下でAndroidManifest.xmlファイルを見つけ、applicationテーブルに次のタグを追加します。



```
<uses-native-library
                android:name="libOpenCL.so"
                android:required="true" />
        //ここでtrueは、このライブラリがないとアプリが正常に動作しないことを意味します。システムに
        //falseは、アプリケーションがこのライブラリ （存在する場合）を使用できますが、特に（必要
        //Android公式サイトの説明：%!s(<nil>)
```

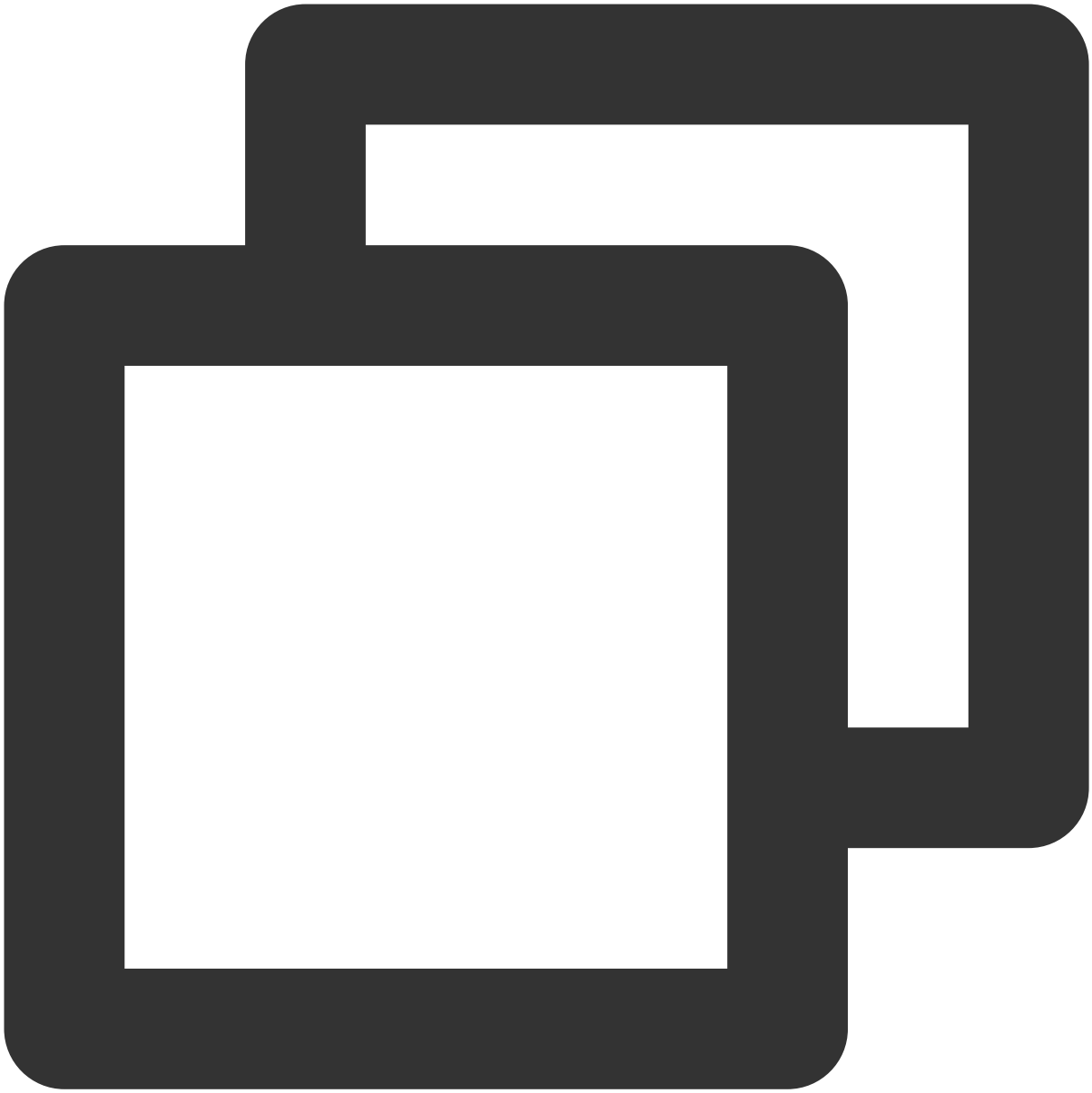
追加すると下図のようになります。



4. 難読化設定

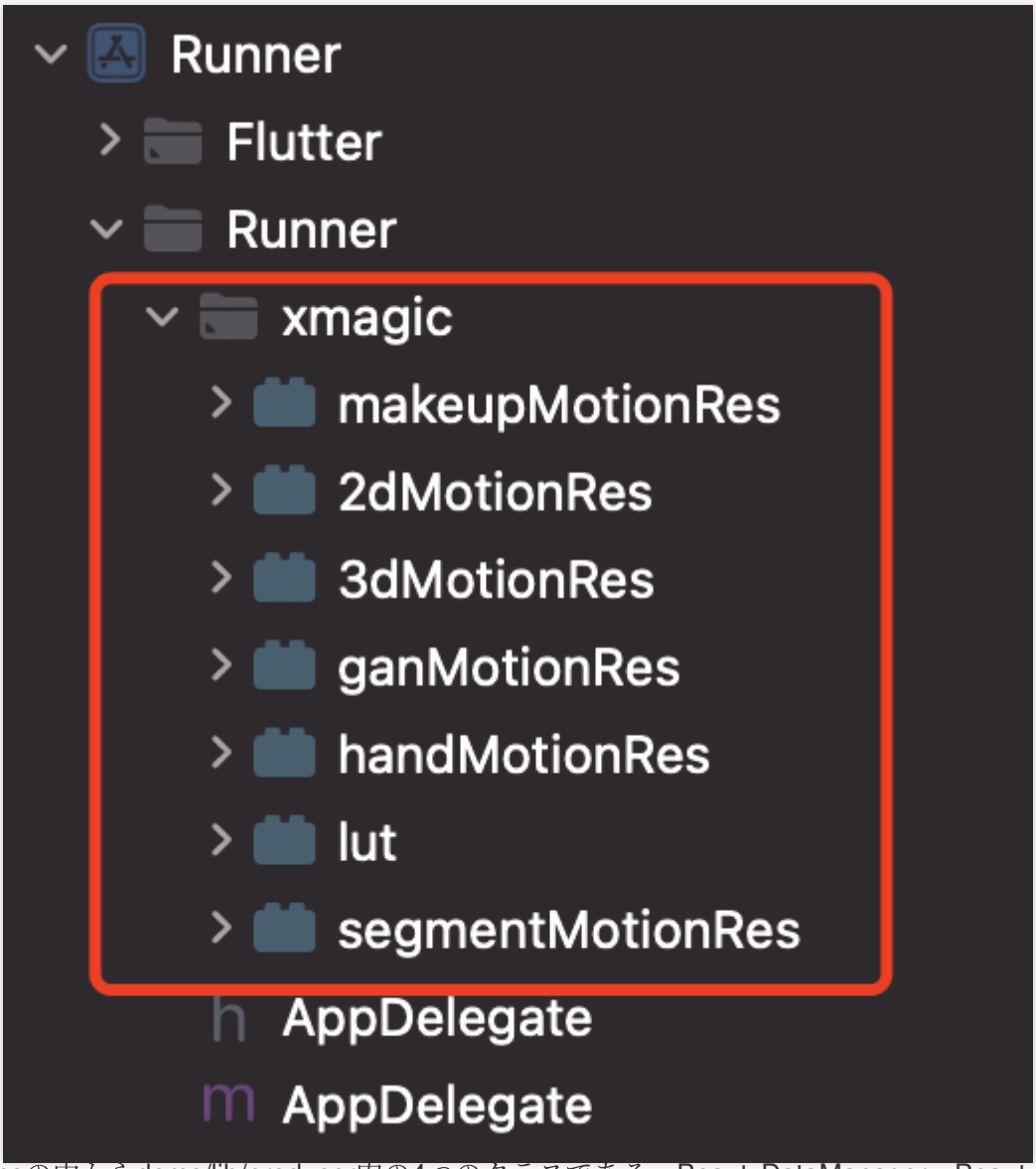
releaseパッケージを作成する際、コンパイルの最適化を有効（minifyEnabledをtrueに設定）にしていると、javaレイヤーで呼び出されないコードがカットされる場合があります。これらのコードはnativeレイヤーで呼び出される場合があり、 `no xxx method` の異常が生じることがあります。

このようなコンパイル最適化を有効にしている場合は、これらのkeepルールを追加し、xmagicのコードがカットされないようにする必要があります。



```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
```

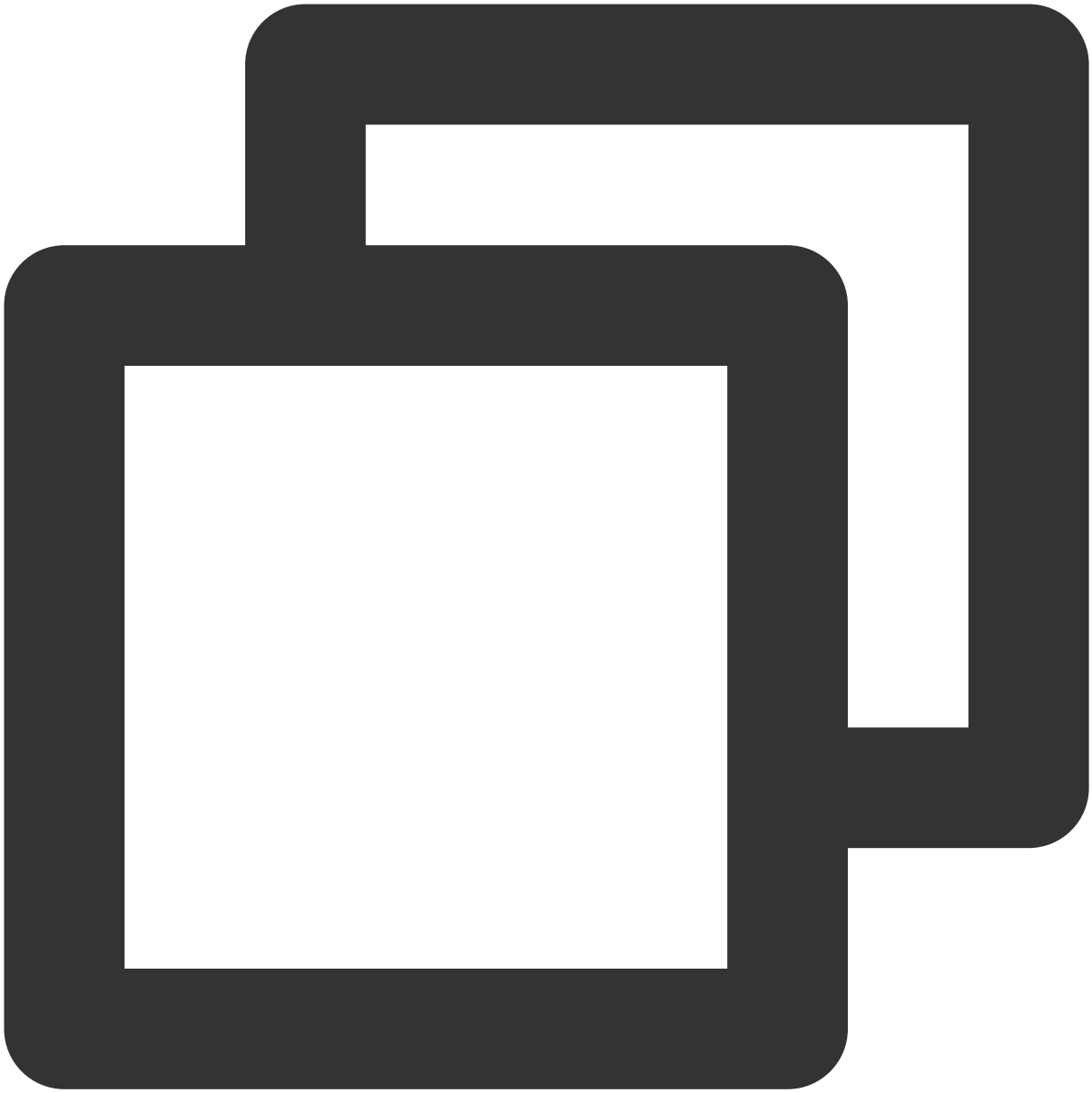
1. **美顔リソースをプロジェクトに追加します**。追加すると下図のようになります（リソースの種類と下図は完全には一致しません）。



2. Demoの中からdemo/lib/producer内の4つのクラスである、BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid、BeautyPropertyProducerIOSをコピーしてご自身のFlutterプロジェクトに追加します。これらの4つのクラスは美顔リソースを設定し、美顔タイプを美顔パネルに表示するために用いられます。

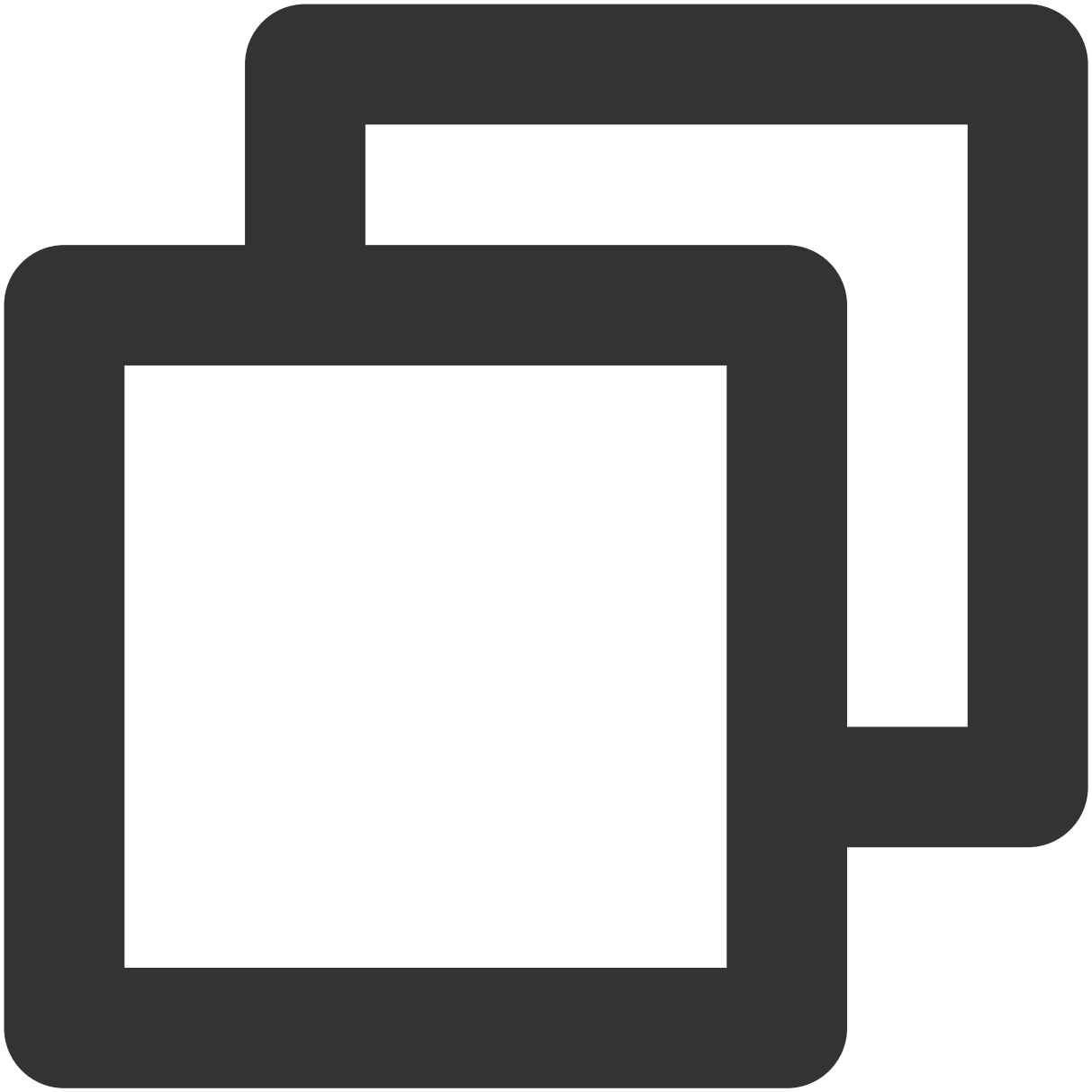
ステップ2：FlutterバージョンSDKの参照

プロジェクトのpubspec.yamlファイルに下記の参照を追加します。



```
tencent_effect_flutter:
    git:
        url: https://github.com/TencentCloud/tencenteffect-sdk-flutter
```

ローカル参照：tencent_effect_flutterから、最新バージョンのtencent_effect_flutterをダウンロードし、フォルダandroid、ios、libとファイルpubspec.yaml、tencent_effect_flutter.imlをプロジェクトディレクトリに追加します。そのあと、プロジェクトのpubspec.yamlファイルの中に以下の参照を追加します。（参考用demo）



```
tencent_effect_flutter:
    path: ../
```

tencent_effect_flutterは、1つのブリッジングヘッダーを提供するのみです。その中で依存するXMagicはデフォルトで最新バージョンになっています。実際に美顔を実現するのはXMagicです。

最新バージョンの美顔SDKを利用される場合は、以下のステップでSDKアップグレードが行えます。

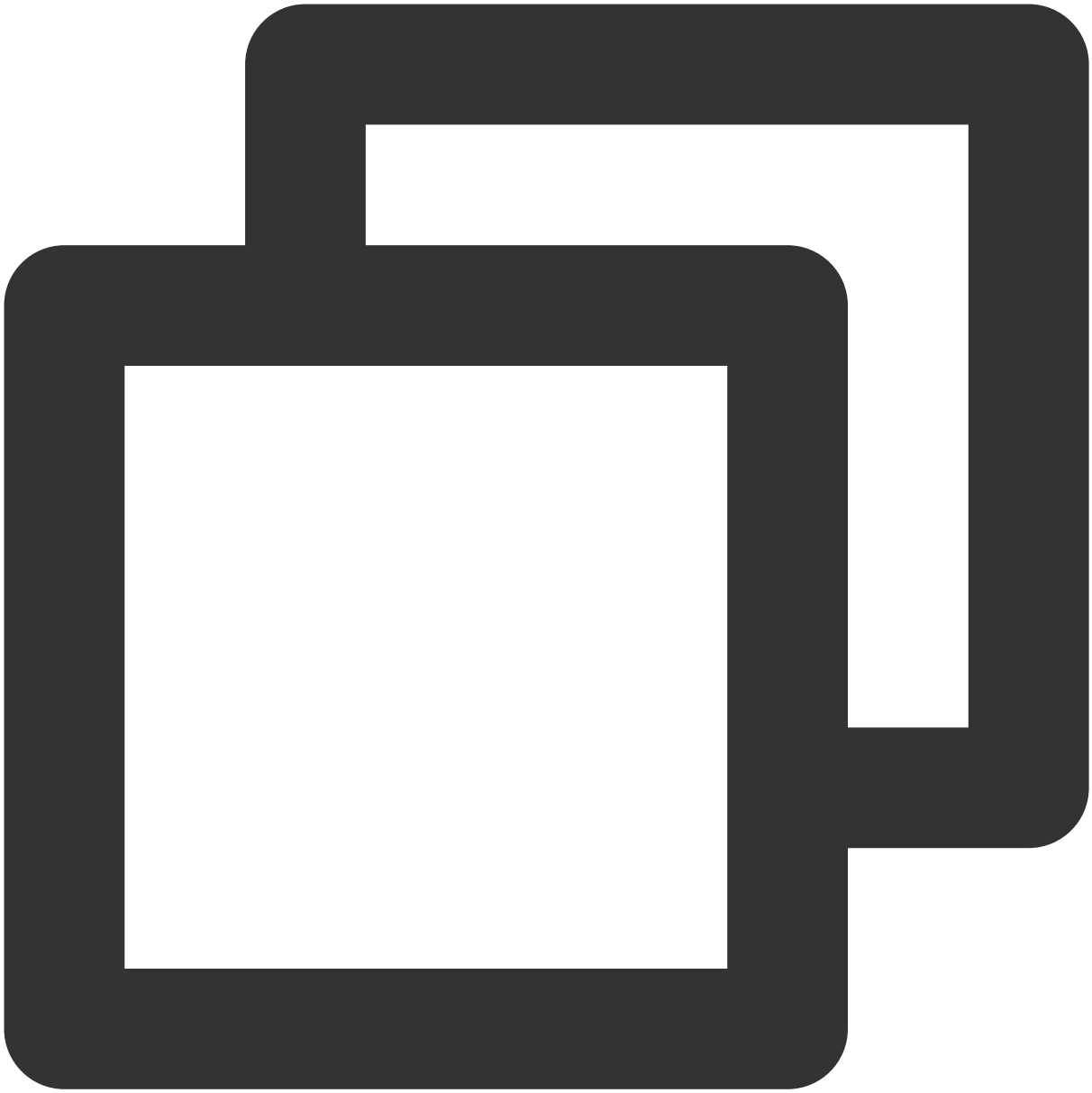
Android

iOS

プロジェクトディレクトリでflutter pub upgradeコマンドを実行、あるいはsubspec.yamlページの右上隅のPub upgradeをクリックします
プロジェクトディレクトリでflutter pub upgradeコマンドを実行、その後iosディレクトリでpod updateコマンドを実行します

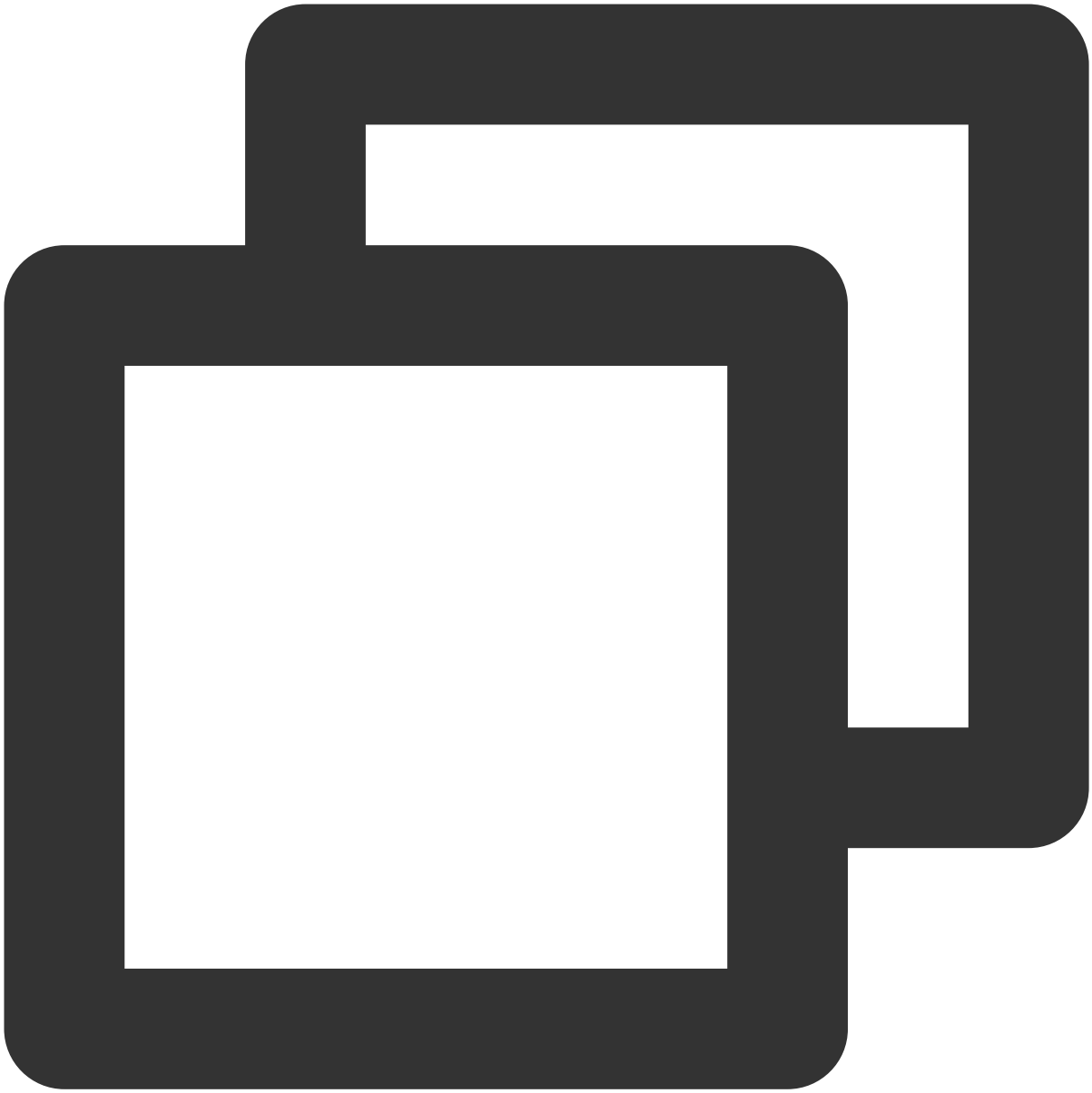
ステップ3：ライブストリーミングとのバインド

Android
iOS
アプリケーションのapplicationクラスのoncreateメソッド（またはFlutterActivityのonCreateメソッド）に次のコードを追加します。



```
TXLivePluginManager.register(new XmagicProcesserFactory());
```

アプリケーションのAppDelegateクラスのdidFinishLaunchingWithOptionsメソッドに次のコードを追加します。

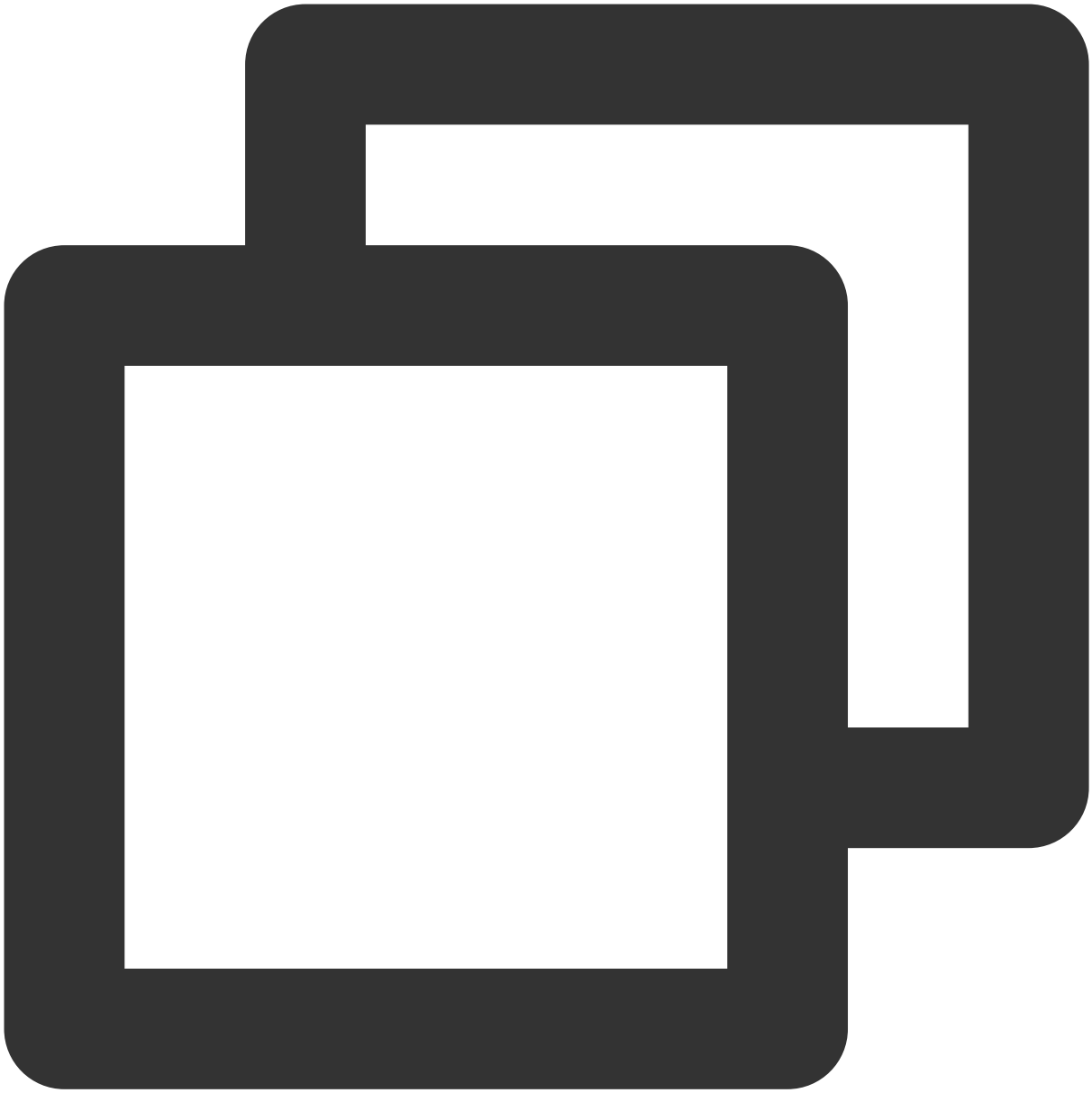


```
XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
[TXLivePluginManager registerWithCustomBeautyProcesserFactory:instance];
```

追加すると下図のようになります。

```objc
#import "AppDelegate.h"
#import "GeneratedPluginRegistrant.h"
@import live_flutter_plugin;
@import tencent_effect_flutter;
@import tencent_trtc_cloud;

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  [GeneratedPluginRegistrant registerWithRegistry:self];
  // Override point for customization after application launch.
  XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
  [TXLivePluginManager registerWithCustomBeautyProcesserFactory:instance];
  [TencentTRTCCloud registerWithCustomBeautyProcesserFactory:instance];
  return [super application:application didFinishLaunchingWithOptions:launchO
}

@end
```

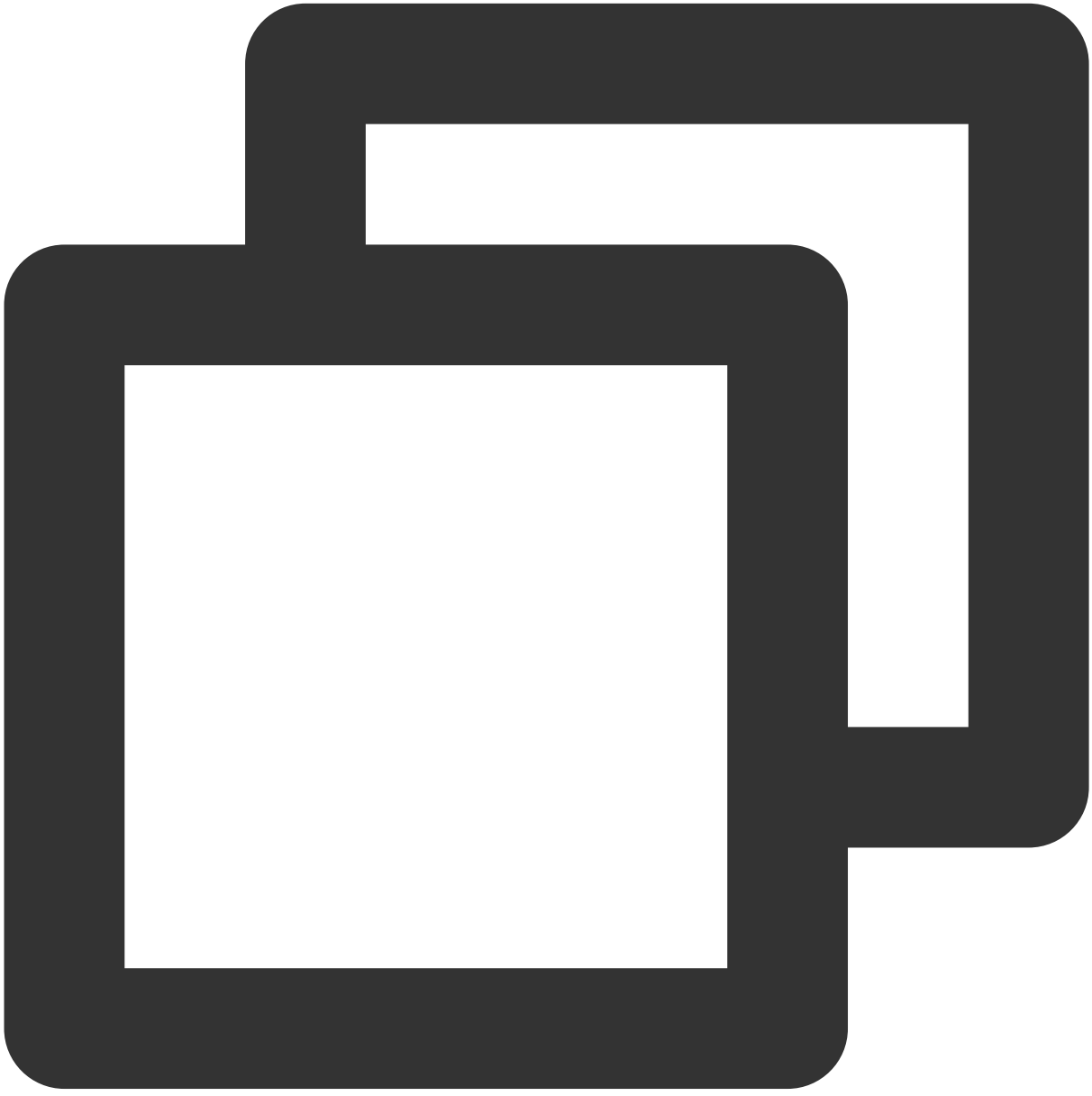
ステップ4：リソース初期化インターフェースの呼び出し



```
String dir =  await BeautyDataManager.getInstance().getResDir();
 TXLog.printlog('ファイルパス：$dir');
 TencentEffectApi.getApi()?.initXmagic(dir,(reslut) {
     _isInitResource = reslut;
     callBack.call(reslut);
     if (!reslut) {
         Fluttertoast.showToast(msg: "リソースの初期化に失敗しました");
     }
 }); TencentEffectApi.getApi()?.initXmagic((reslut) {
     if (!reslut) {
         Fluttertoast.showToast(msg: "リソースの初期化に失敗しました");
```

```
        }
    });
```

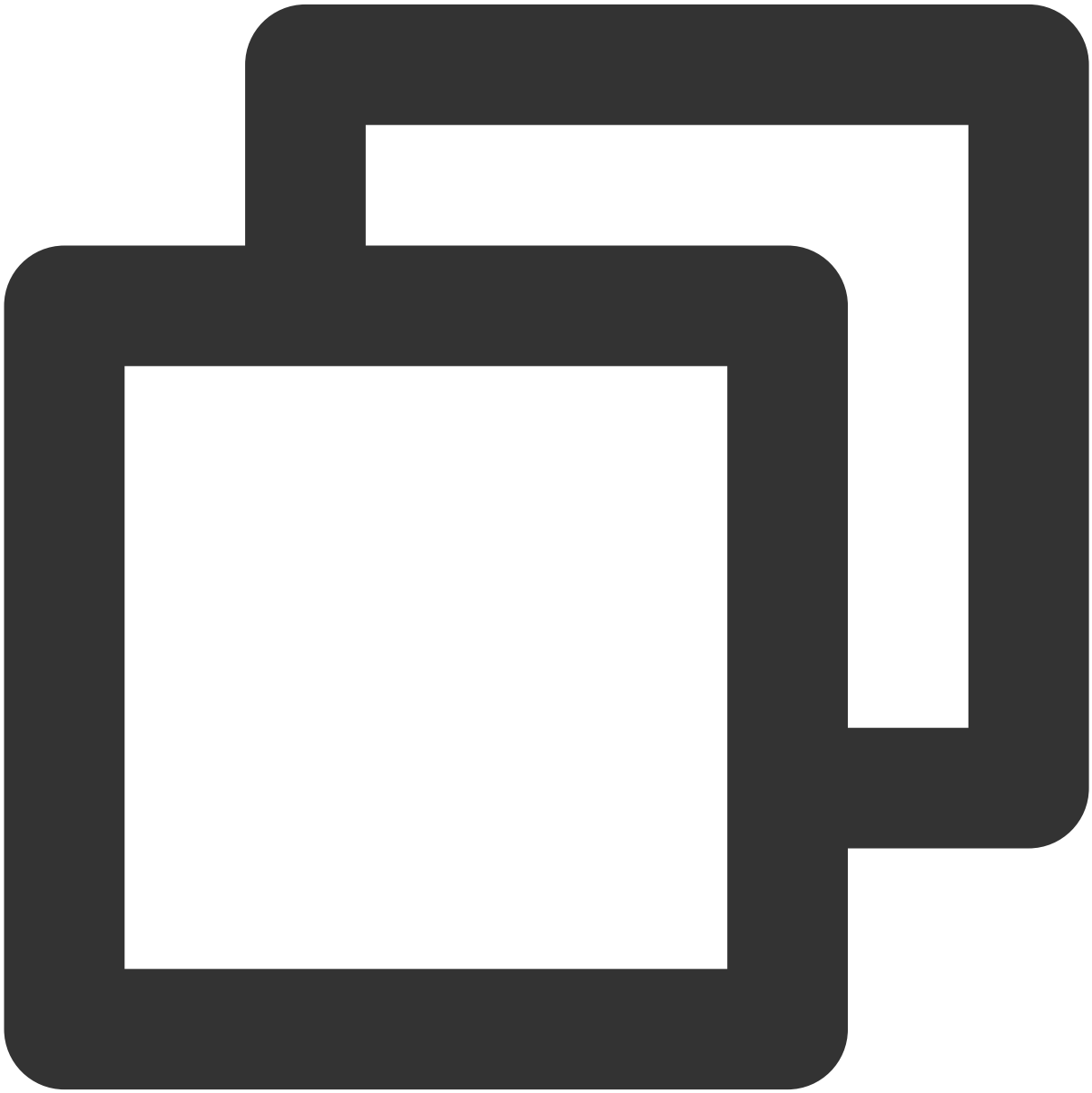
ステップ5：美顔権限承認を実行



```
TencentEffectApi.getApi()?.setLicense(licenseKey, licenseUrl,
                                      (errorCode, msg) {
                                              TXLog.printlog("認証結果を印刷します errorC
```

```
                                                    if (errorCode == 0) {
                                                        //認証に成功しました
                                                    }
    });
```

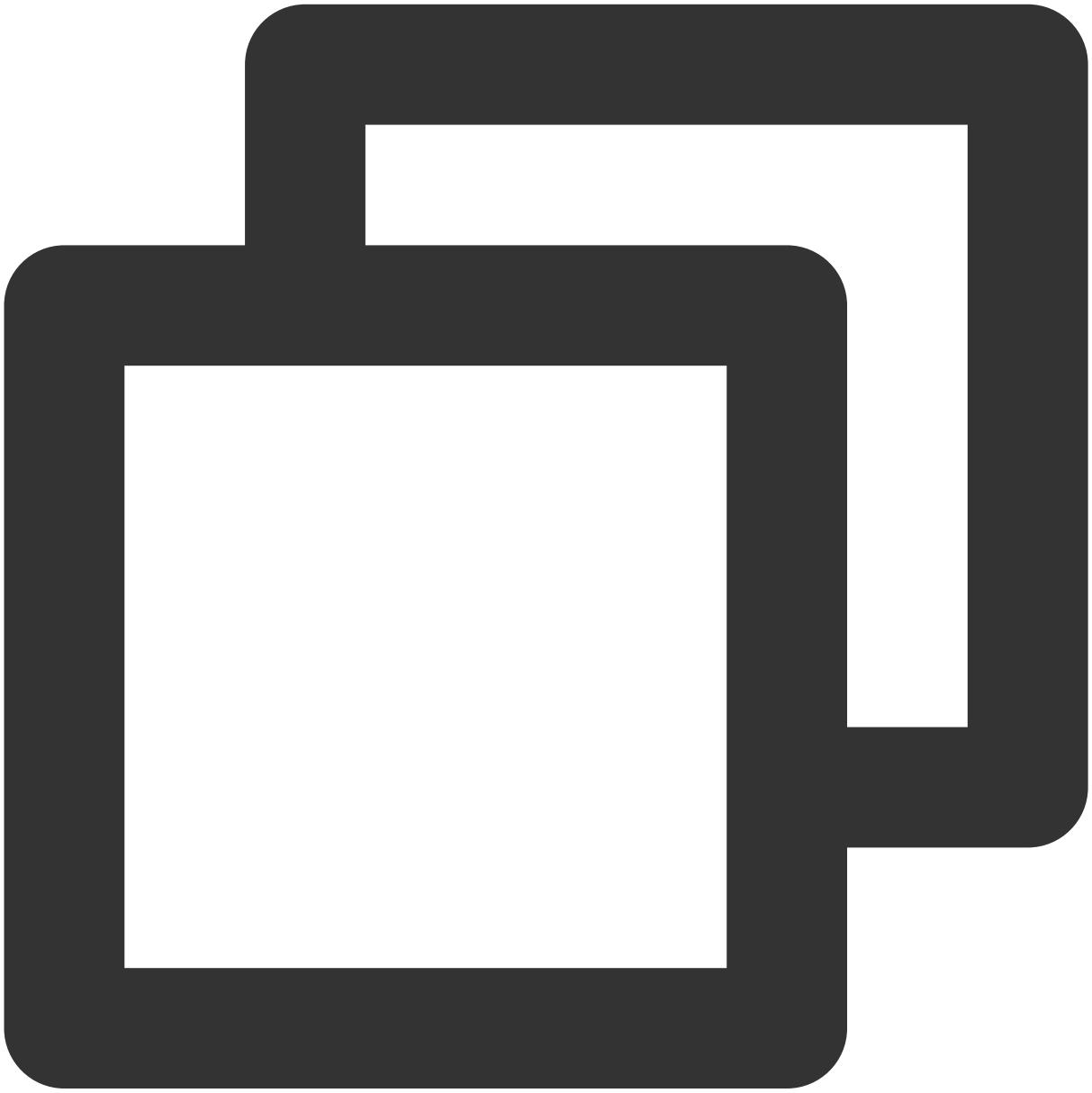
ステップ6：美顔を有効にする



```
///美顔操作を有効にします
```

```
var enableCustomVideo = await _livePusher?.enableCustomVideoProcess(true);
```

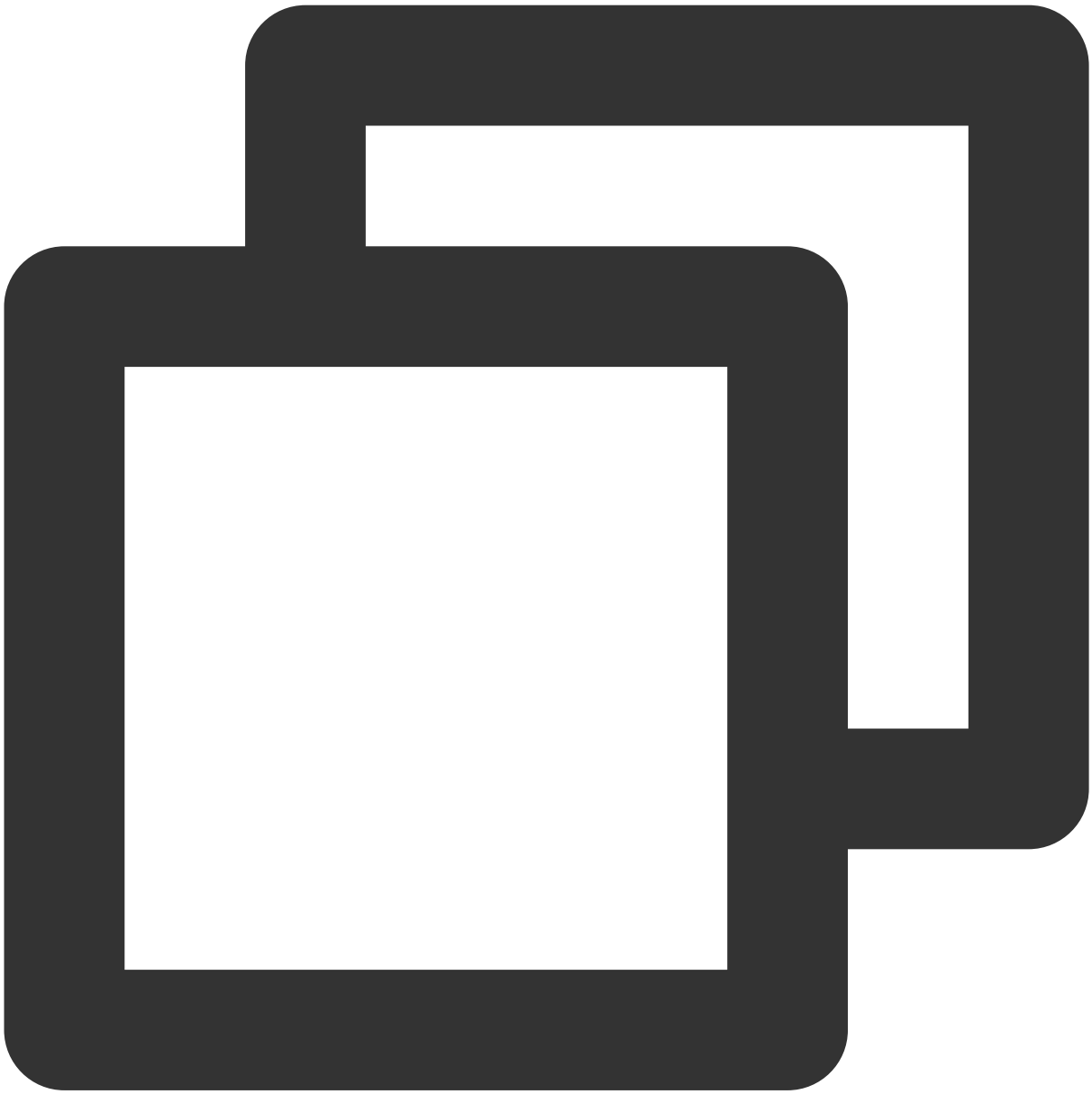
ステップ7：美顔属性の設定



```
TencentEffectApi.getApi()?.updateProperty(_xmagicProperty!);
///_xmagicPropertyではBeautyDataManager.getInstance().getAllPannelData();によってすべ
```

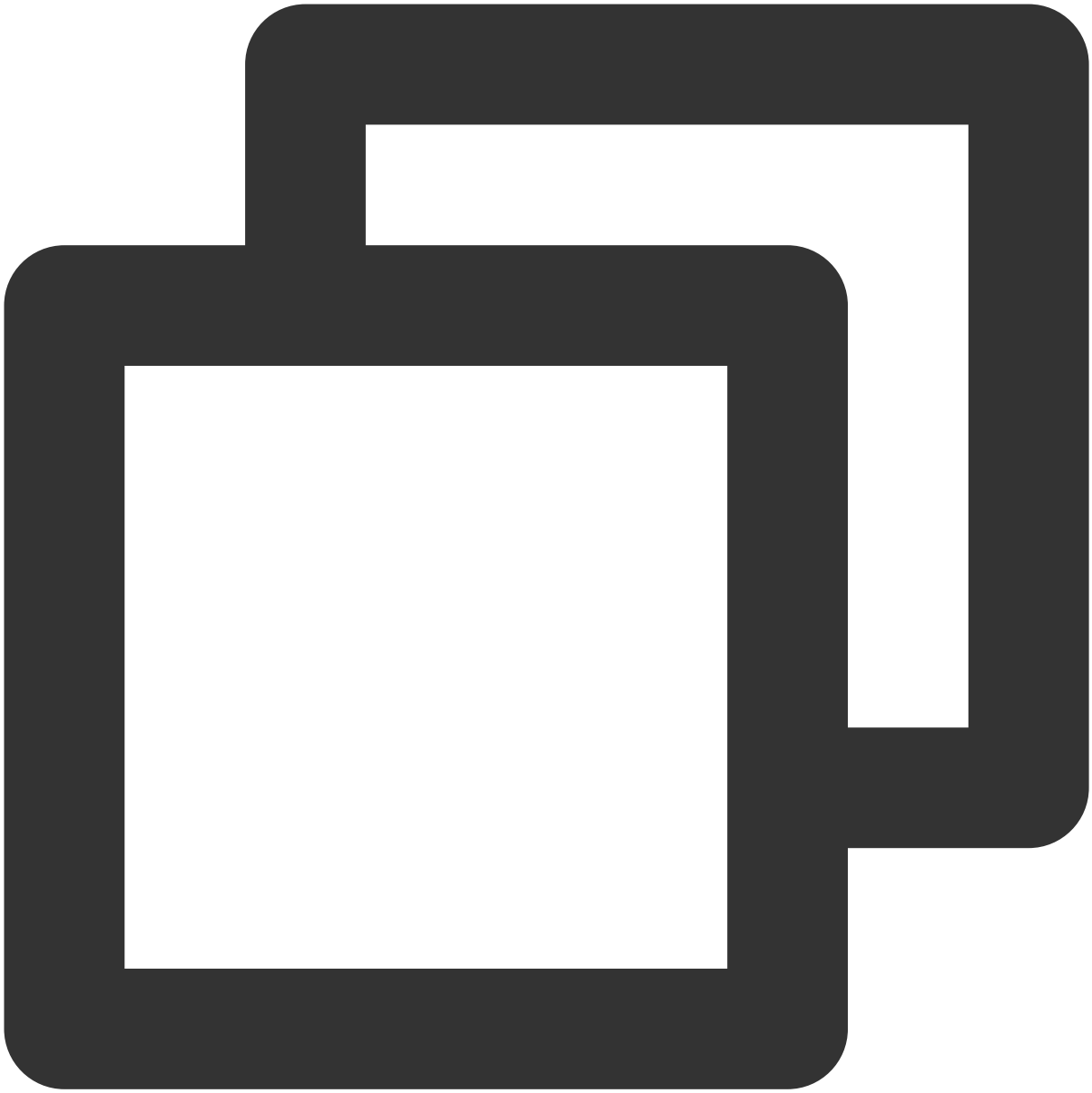
ステップ8：その他の属性の設定

美顔サウンドエフェクトの一時停止



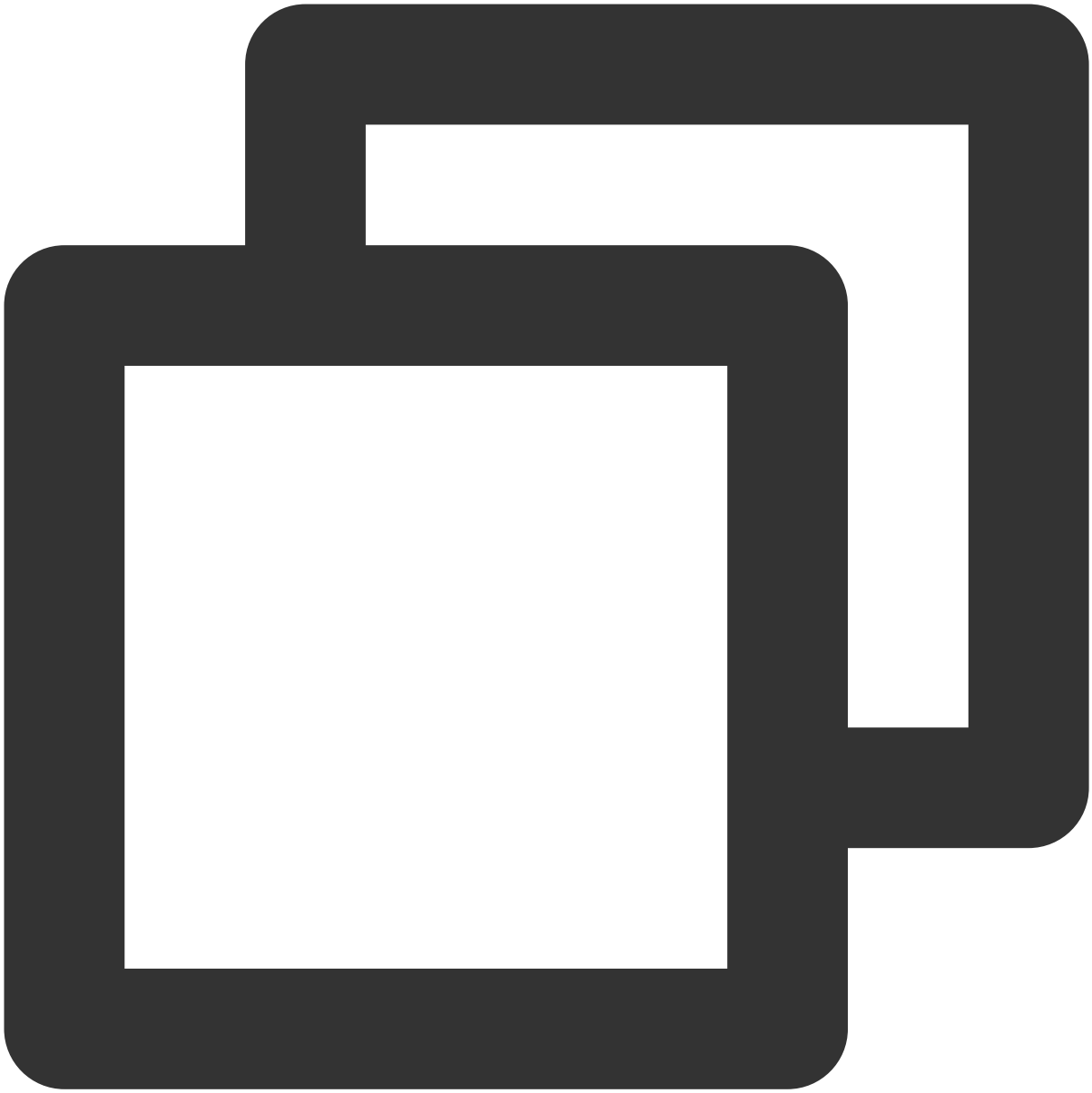
```
TencentEffectApi.getApi()?.onPause();
```

美顔サウンドエフェクトの再開



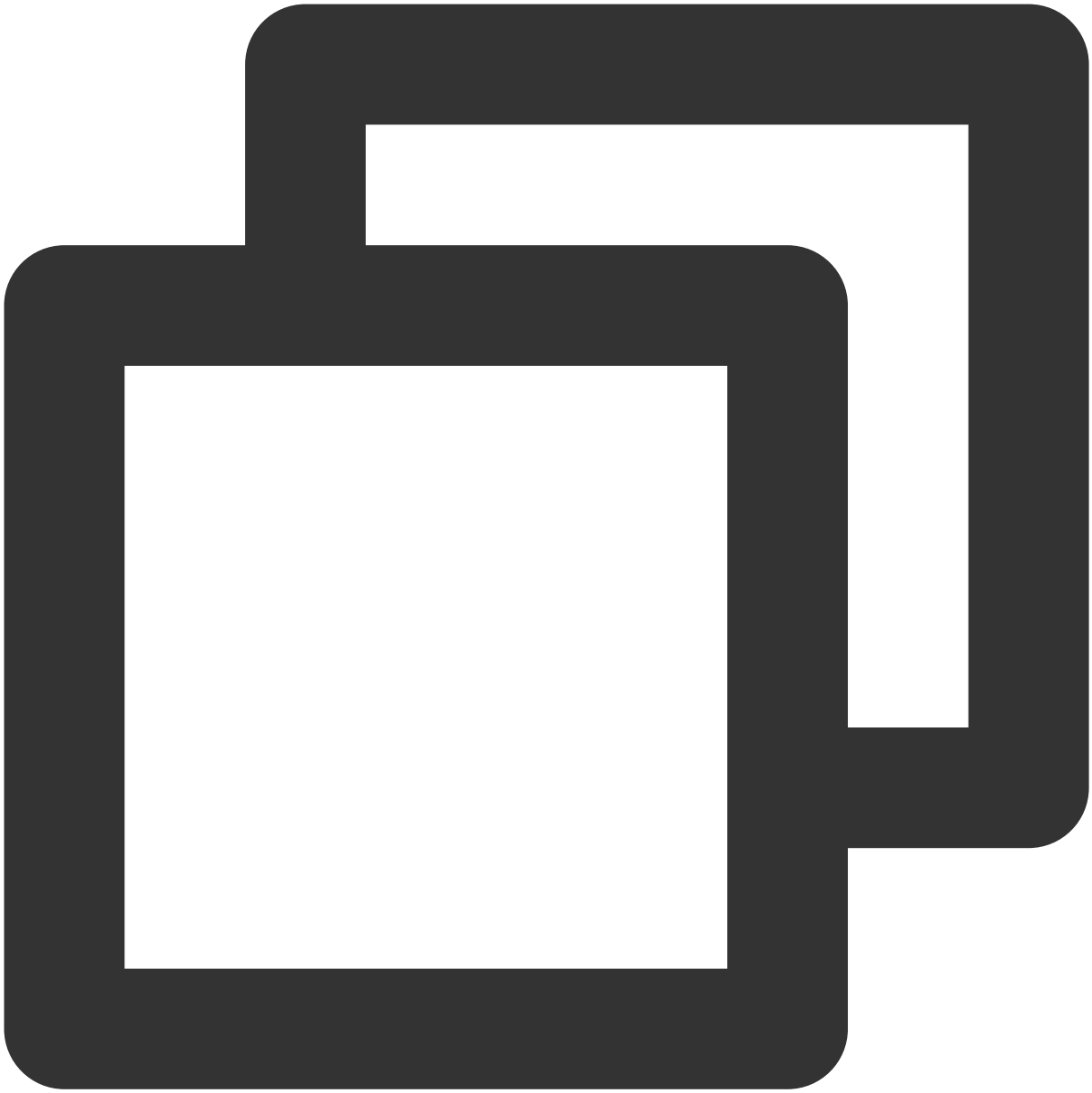
```
TencentEffectApi.getApi()?.onResume();
```

美顔イベントの監視



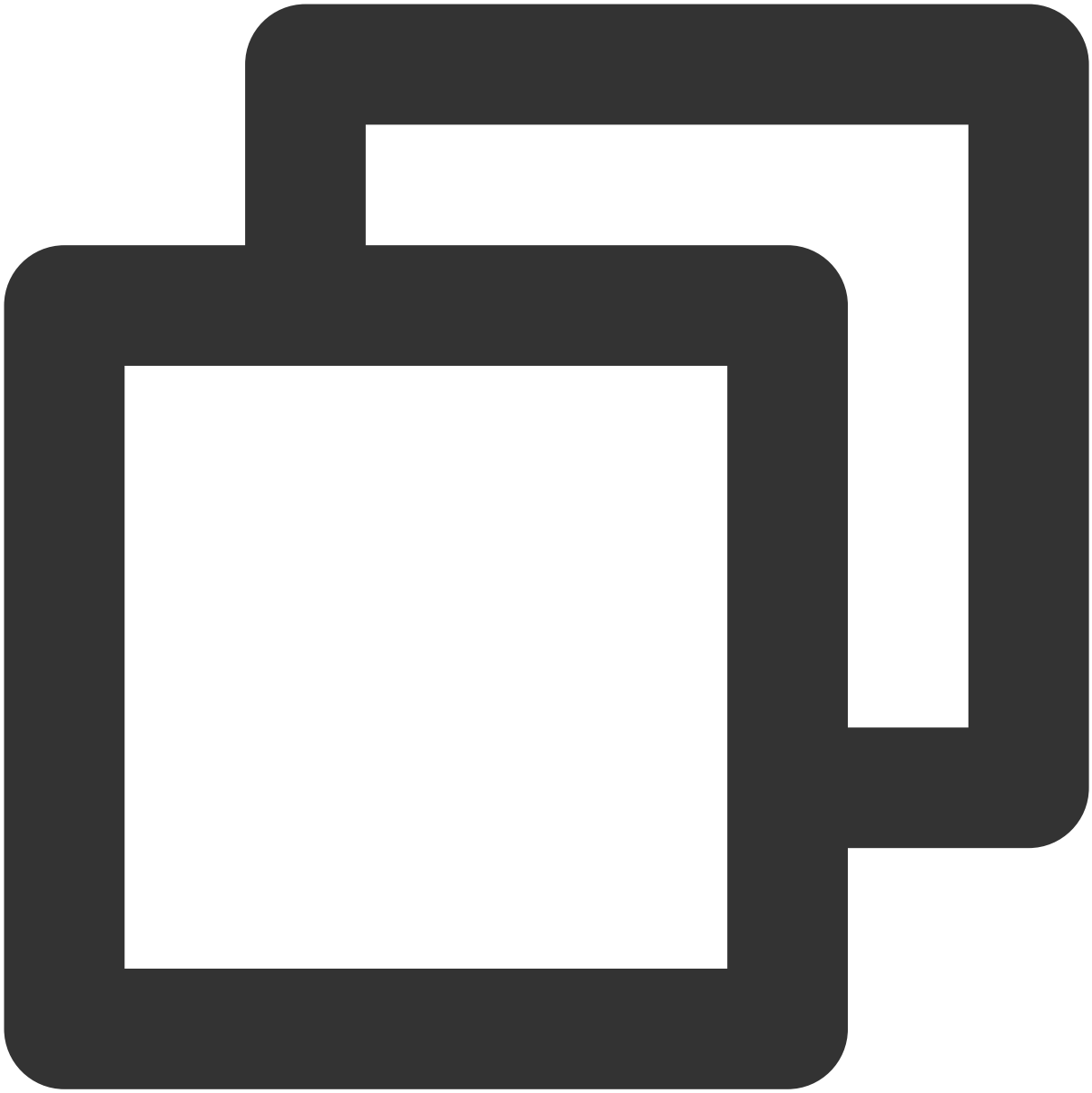
```
TencentEffectApi.getApi()
            ?.setOnCreateXmagicApiErrorListener((errorMsg, code) {
                TXLog.printlog("美顔オブジェクトの作成にエラーが発生しました errorMsg = $er
    });    ///美顔を作成する前に設定が必要です
```

顔、ジェスチャー、体の検出状態コールバックの設定



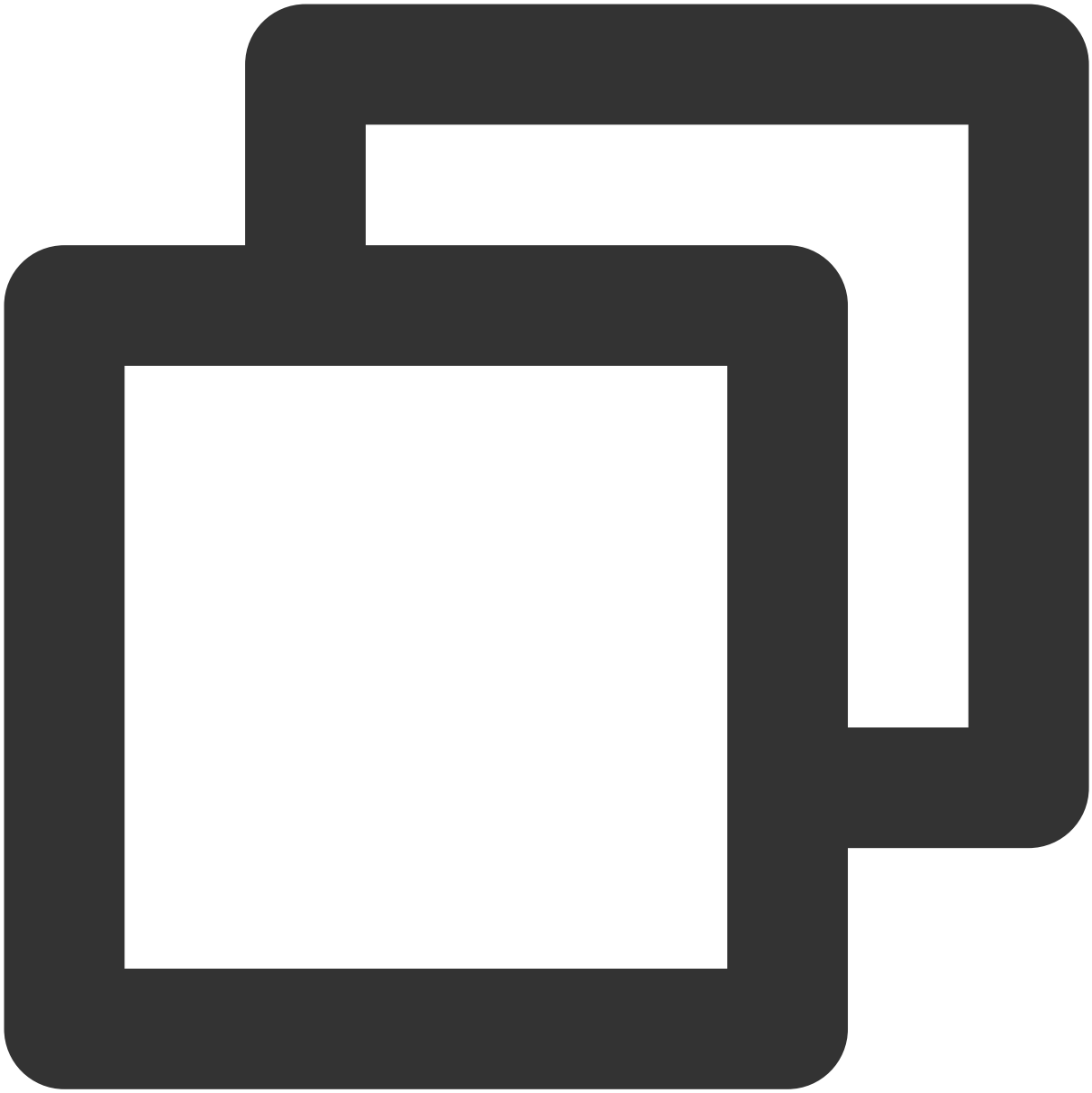
```
TencentEffectApi.getApi()?.setAIDataListener(XmagicAIDataListenerImp());
```

動的エフェクトプロンプトのコールバック関数の設定



```
TencentEffectApi.getApi()?.setTipsListener(XmagicTipsListenerImp());
```

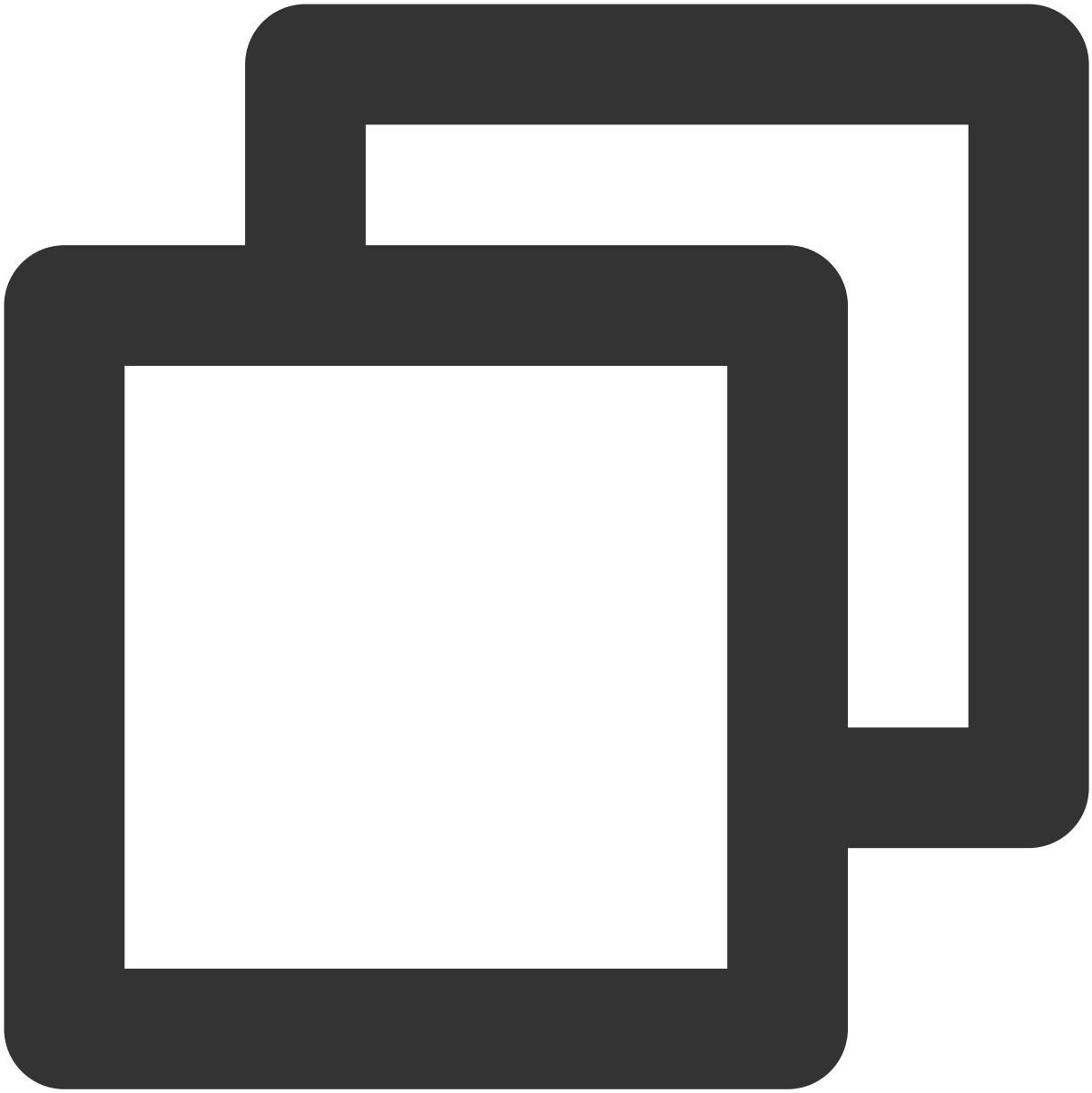
顔の特徴点位置情報などのデータコールバックを設定します（S1-05およびS1-06パッケージのみコールバックあり）



```
TencentEffectApi.getApi()?.setYTDataListener((data) {
       TXLog.printlog("setYTDataListener  $data");
   });
```

すべてのコールバックの削除

ページが破棄された場合はすべてのコールバックを削除する必要があります。



```
TencentEffectApi.getApi()?.setOnCreateXmagicApiErrorListener(null);
TencentEffectApi.getApi()?.setAIDataListener(null);
TencentEffectApi.getApi()?.setYTDataListener(null);
TencentEffectApi.getApi()?.setTipsListener(null);
```

説明

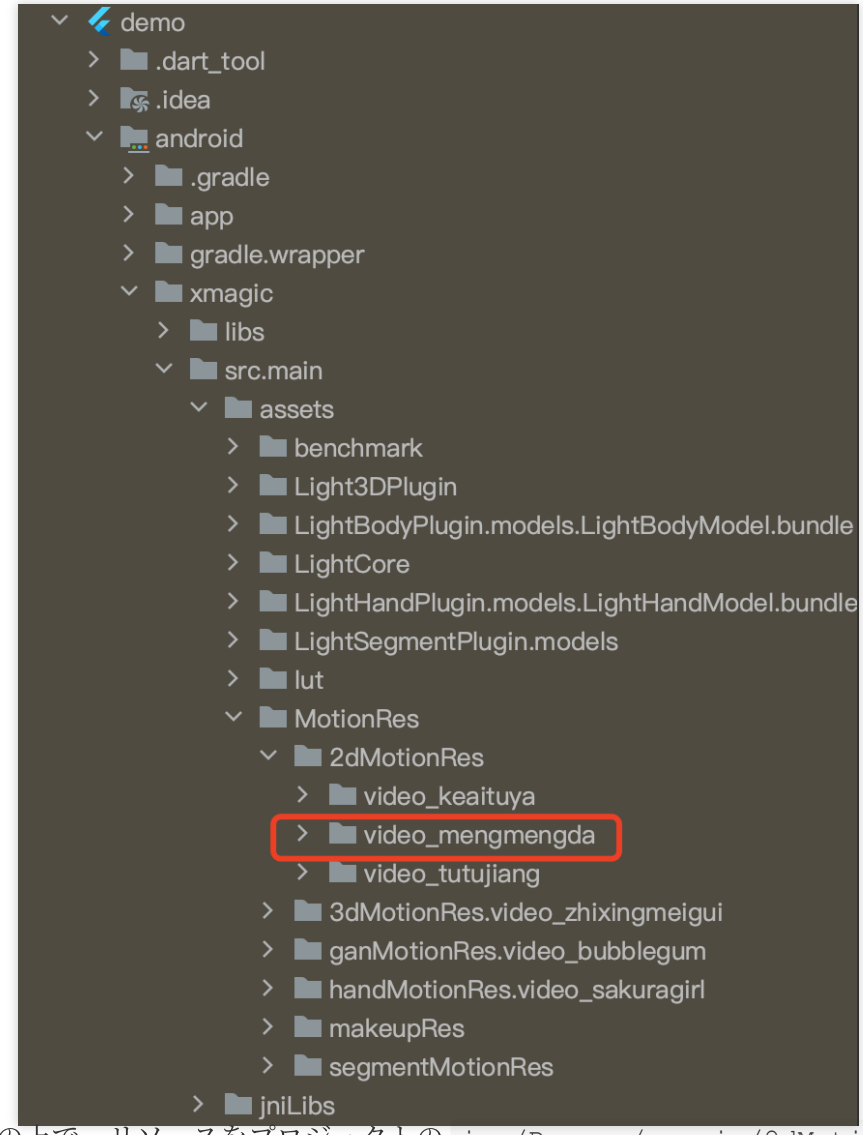
インターフェースの詳細についてはインターフェースドキュメントを、その他についてはDemoプロジェクトをそれぞれ参照できます。

ステップ9：美顔パネル上の美顔データの追加と削除

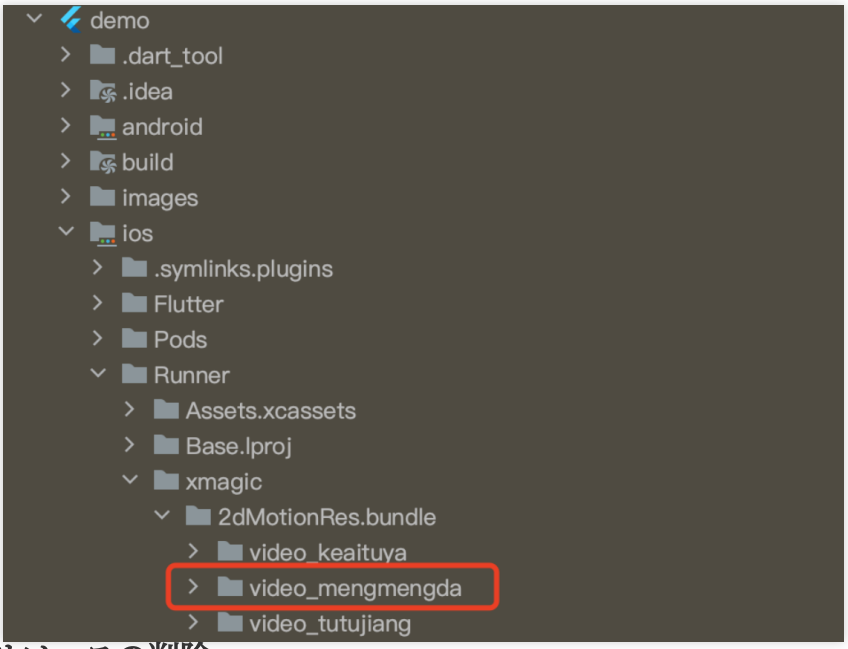
BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid、BeautyPropertyProducerIOSの4つのクラスでは、美顔パネルのデータ設定を自由に操作できます。

美顔リソースの追加

1. ステップ1の方法に従ってリソースファイルを対応するリソースフォルダ内に追加します。例えば、2D動的エフェクトのリソースを追加したい場合は、リソースをプロジェクトの `android/xmagic/src.mian/assets/MotionRes/2dMotionRes` ディレクトリ下に置く必要があります。



2. その上で、リソースをプロジェクトの `ios/Runner/xmagic/2dMotionRes.bundle` ディレクトリ下に追加します。



美顔リソースの削除

Licenseによっては美顔と美ボディの一部機能の権限がない場合があります。この一部機能は美顔パネル上に表示する必要はないため、美顔パネルデータの設定でこの一部機能の設定を削除する必要があります。例えばリップのエフェクトを削除する場合は、
BeautyPropertyProducerAndroidクラスおよびBeautyPropertyProducerIOSクラスのgetBeautyDataメソッドから次のコードをそれぞれ削除します。

```
// Map<String, String> lipsResPathNames = {};
// lipsResPathNames["lips_fuguhong.png"] = "复古红";
// lipsResPathNames["lips_mitaose.png"] = "蜜桃色";
// lipsResPathNames["lips_shanhuju.png"] = "珊瑚橘";
// lipsResPathNames["lips_wenroufen.png"] = "温柔粉";
// lipsResPathNames["lips_huolicheng.png"] = "活力橙";
// List<XmagicUIProperty> itemLipsPropertys = [];
// String lipId = "beauty.lips.lipsMask";
// for (String ids in lipsResPathNames.keys) {
//   itemLipsPropertys.add(XmagicUIProperty(
//       uiCategory: Category.BEAUTY,
//       displayName: lipsResPathNames[ids]!,
//       id: lipId,
//       resPath: resPaths + ids,
//       thumbDrawableName: "beauty_lips",
//       effKey: BeautyConstant.BEAUTY_MOUTH_LIPSTICK,
//       effValue: XmagicPropertyValues(0, 100, 50, 0, 1),
//       rootDisplayName: "口红"));
// }
// XmagicUIProperty itemLips = XmagicUIProperty(
//     displayName: "口红",
//     thumbDrawableName: "beauty_lips",
//     uiCategory: Category.BEAUTY);
// itemLips.xmagicUIPropertyList = itemLipsPropertys;
// beautyList.add(itemLips);
```

Tencent EffectをTRTC SDKに統合
ios

最終更新日：：2023-02-27 14:18:15

統合の準備

1. Demoパッケージをダウンロードして解凍し、Demoプロジェクトのxmagicモジュール（bundle、XmagicIconRes、Xmagicフォルダ）を実際のプロジェクトにインポートします。
2. **使用しているXMagic SDKのバージョンが2.5.0より古い場合は、**SDKディレクトリ内の `libpag.framework` 、 `Masonry.framework` 、 `XMagic.framework` 、 `YTCommonXMagic.framework` をインポートします。**使用しているXMagic SDKのバージョンが2.5.1およびそれ以降の場合は、**SDKディレクトリ内の `libpag.framework` 、 `Masonry.framework` 、 `XMagic.framework` 、 `YTCommonXMagic.framework` 、 `Audio2Exp.framework` 、 `TEFFmpeg.framework` をインポートします。
3. frameworkの署名は、**General--> Masonry.framework**と**libpag.framework**は**Embed & Sign**を選択します。**YTCommonXMagic.framework** は、バージョン2.5.1より古い場合は**Do Not Embed**を選択し、バージョン2.5.1およびそれ以降の場合は**Embed & Sign**を選択します。
4. Bundle IDを、テスト用に申請した権限と同じものに変更します。

開発者環境要件

開発ツールXCode 11およびそれ以上：App Storeまたはダウンロードアドレスをクリックします。
推奨実行環境：
デバイス要件：iPhone 5およびそれ以上である必要があります。iPhone 6およびそれ未満の場合は、フロントカメラのサポートを最大720pとし、1080pはサポートしていません。
システム要件：iOS 10.0およびそれ以降のバージョン。

C/C++レイヤー開発環境

XCodeはデフォルトではC++環境となります。

タイプ	依存ライブラリ
システム依存ライブラリ	Accelerate AssetsLibrary AVFoundation CoreMedia CoreFoundation CoreML

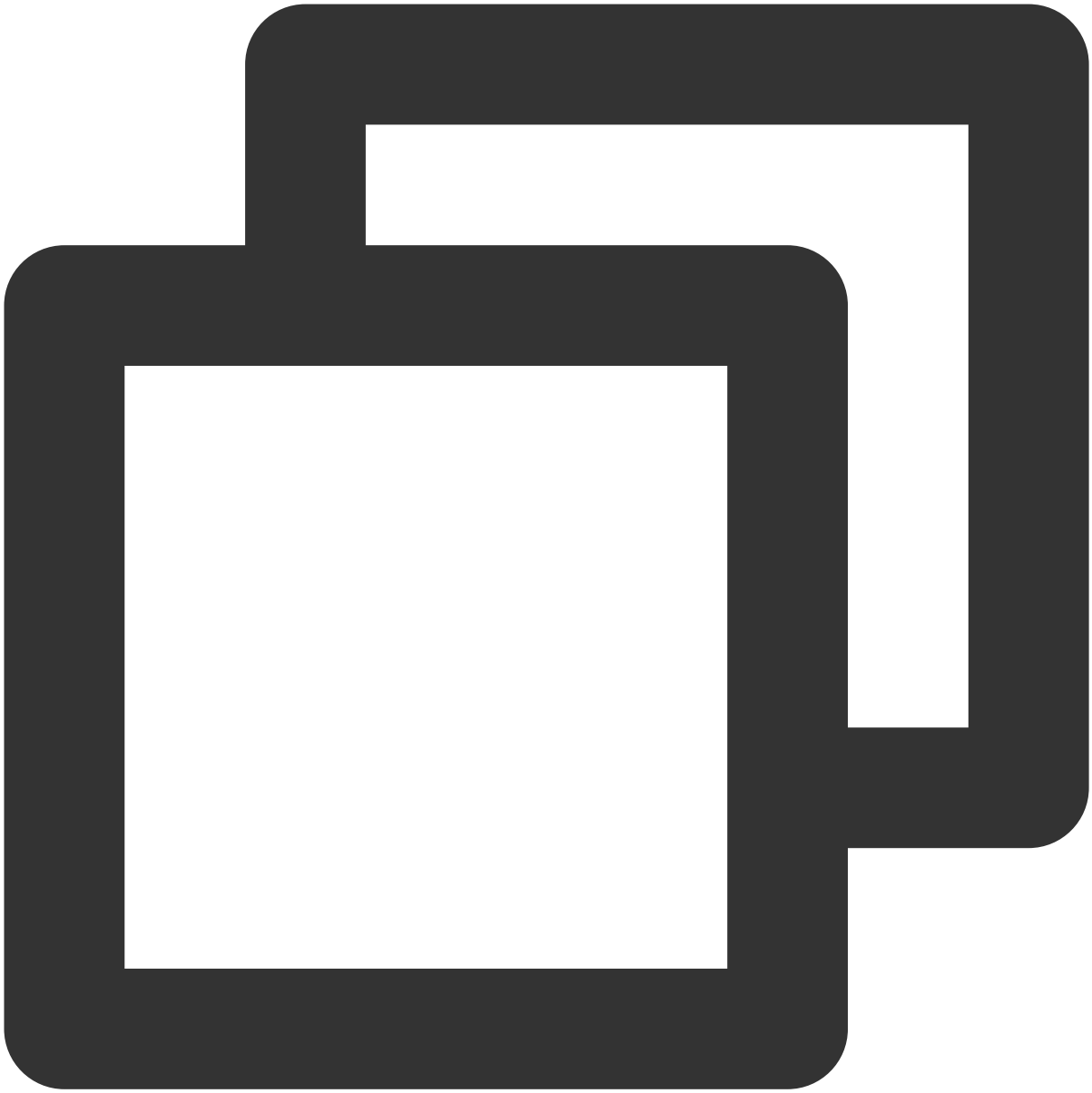
	Foundation JavaScriptCore libc++.tbd libz.b libresolv.tbd libsqlite3.0.tbd MetalPerformanceShaders MetalKit MobileCoreServices OpneAL OpneGLES Security ReplayKit SystemConfiguration UIKit
付属ライブラリ	YTCommon（認証静的ライブラリ） XMagic（美顔静的ライブラリ） libpag（ビデオデコード動的ライブラリ） Masonry（ウィジェットレイアウトライブラリ） TXLiteAVSDK_Professional TXFFmpeg TXSoundTouch Audio2Exp（xmagic sdk versionは2.5.1およびそれ以降のバージョンのみ） TEFFmpeg（xmagic sdk versionは2.5.1およびそれ以降のバージョンのみ）

SDKインターフェースの統合

ステップ1およびステップ2については、Demoプロジェクトの `ThirdBeautyViewController` クラスの `viewDidLoad` 、 `buildBeautySDK` メソッドを参照できます。 `AppDelegate` クラスの `application` メソッドはXmagic認証を実行します。
ステップ4からステップ7までは、Demoプロジェクトの `ThirdBeautyViewController` 、 `BeautyView` クラスの関連インスタンスコードを参照できます。

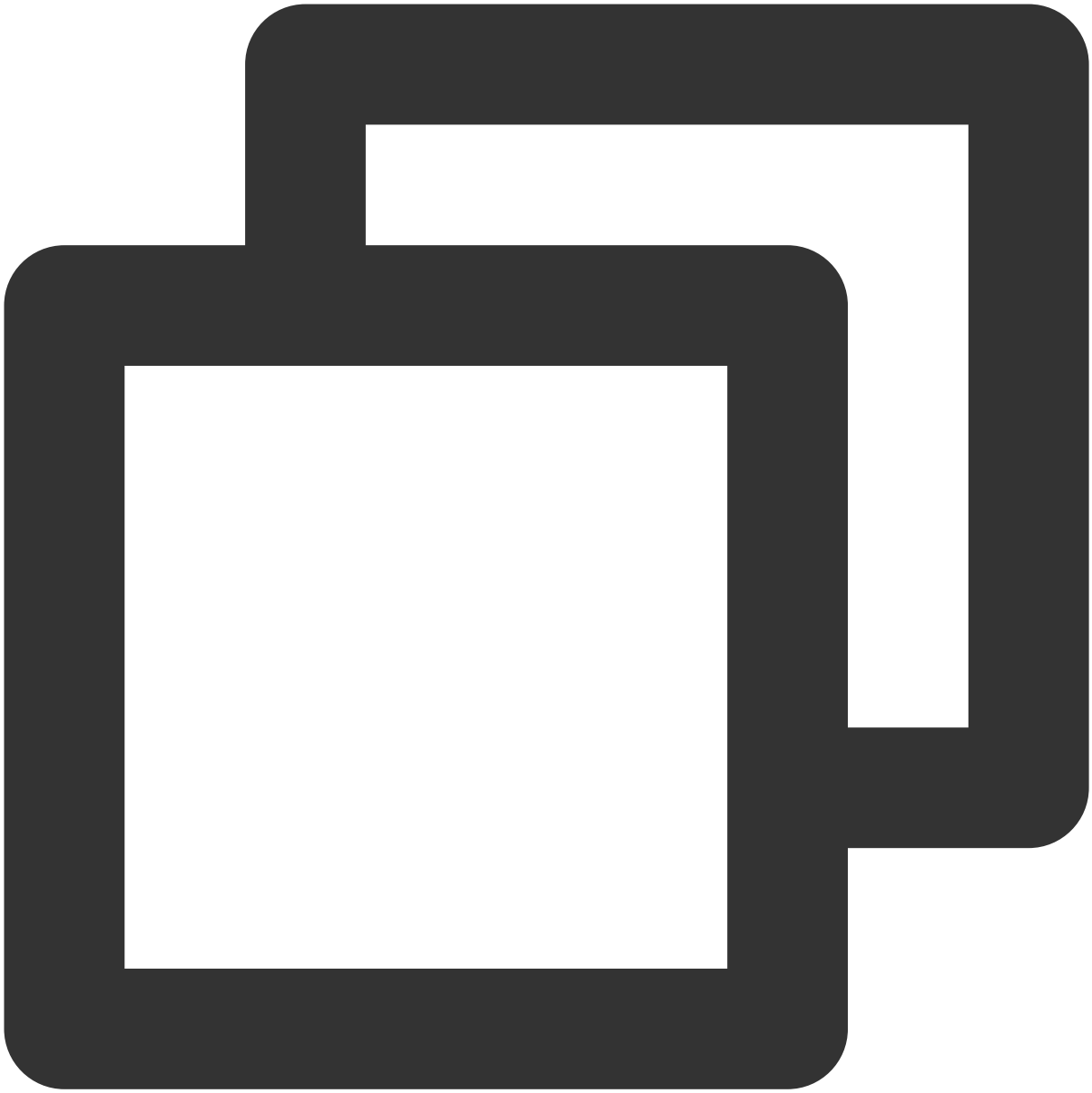
ステップ1：権限の初期化

1. 初めにプロジェクトの `AppDelegate` の `didFinishLaunchingWithOptions` に次の認証コードを追加します。このうち `LicenseURL` と `LicenseKey` はTencent Cloud公式サイトに権限承認を申請した際の情報とします。



```
[TXLiveBase setLicenceURL:LicenseURL key:LicenseKey];
```

2. Xmagicの認証：関連業務モジュールの初期化コードの中でURLとKEYを設定し、licenseのダウンロードをトリガーします。使用する直前になってダウンロードすることは避けてください。あるいは `AppDelegate` の `didFinishLaunchingWithOptions` メソッドでダウンロードをトリガーすることもできます。このうち、 `LicenseURL` と `LicenseKey` はコンソールでLicenseをバインドした際に生成された権限承認情報です。SDKのバージョンが2.5.1より古い場合、 `TELicenseCheck.h` は `XMagic.framework` 内にあります。SDKのバージョンが2.5.1およびそれ以降の場合、 `TELicenseCheck.h` は `YTCommonXMagic.framework` 内にあります。



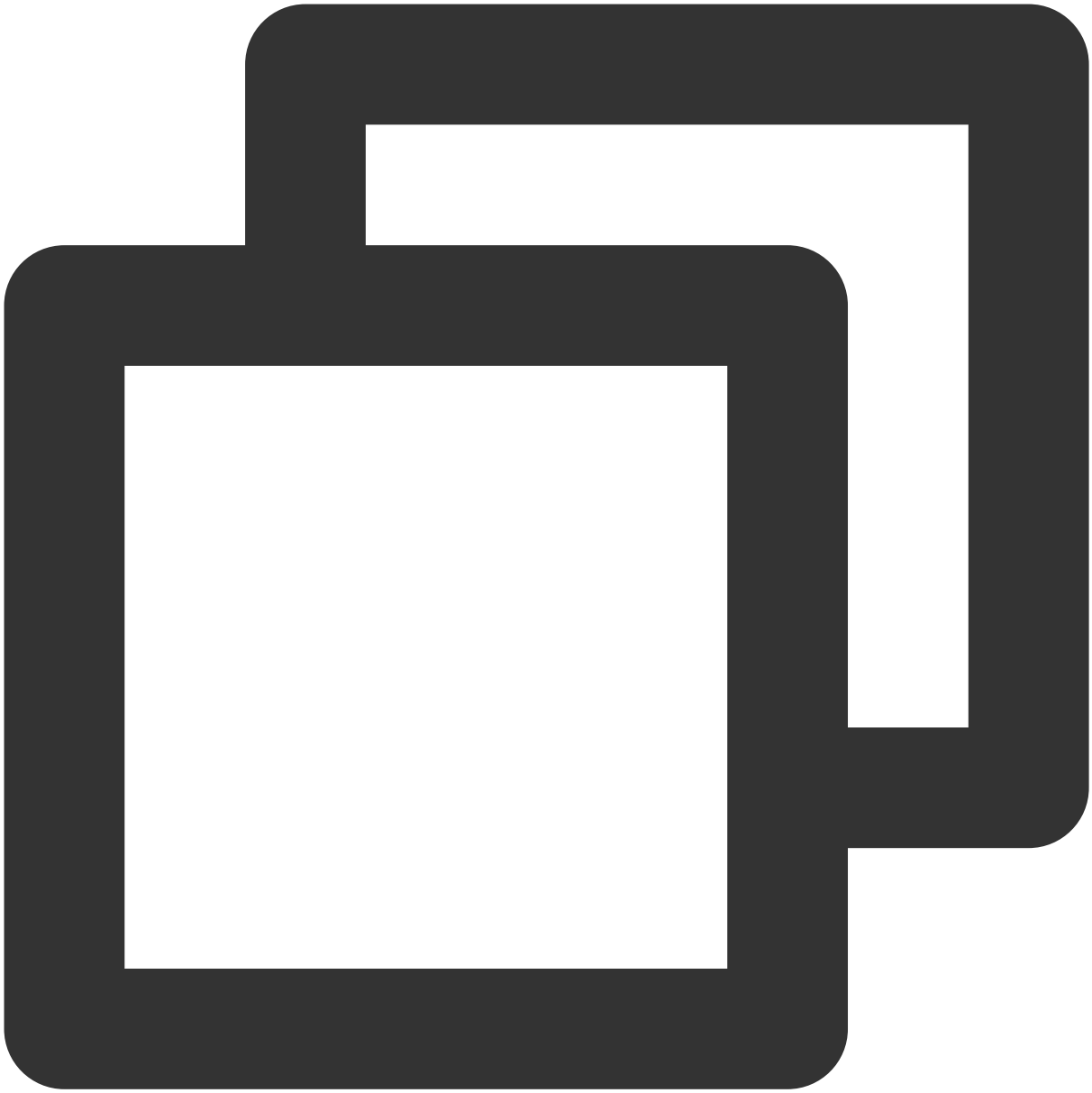
```
[TELicenseCheck setTELicense:LicenseURL key:LicenseKey completion:^(NSInteger authr
    if (authresult == TELicenseCheckOk) {
        NSLog(@"認証成功");
    } else {
        NSLog(@"認証失敗");
    }
}];
```

認証errorCode説明：

エラーコード	説明

0	成功です。Success
-1	入力パラメータが無効です（例：URLまたはKEYが空など）
-3	ダウンロードの段階で失敗しました。ネットワークの設定を確認してください
-4	ローカルから読み取ったTE権限承認情報が空です。IOの失敗による可能性があります
-5	読み取ったVCUBE TEMP Licenseファイルの内容が空です。IOの失敗による可能性があります
-6	v_cube.licenseファイルのJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-7	署名の検証に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-8	復号に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-9	TELicenseフィールド内のJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-10	ネットワークから解析したTE権限承認情報が空です。Tencent Cloudチームに連絡して処理を依頼してください
-11	TE権限承認情報をローカルファイルに書き込む際に失敗しました。IOの失敗による可能性があります
-12	ダウンロードに失敗しました。ローカルassetの解析も失敗しました
-13	認証に失敗しました
その他	Tencent Cloudチームに連絡して処理を依頼してください

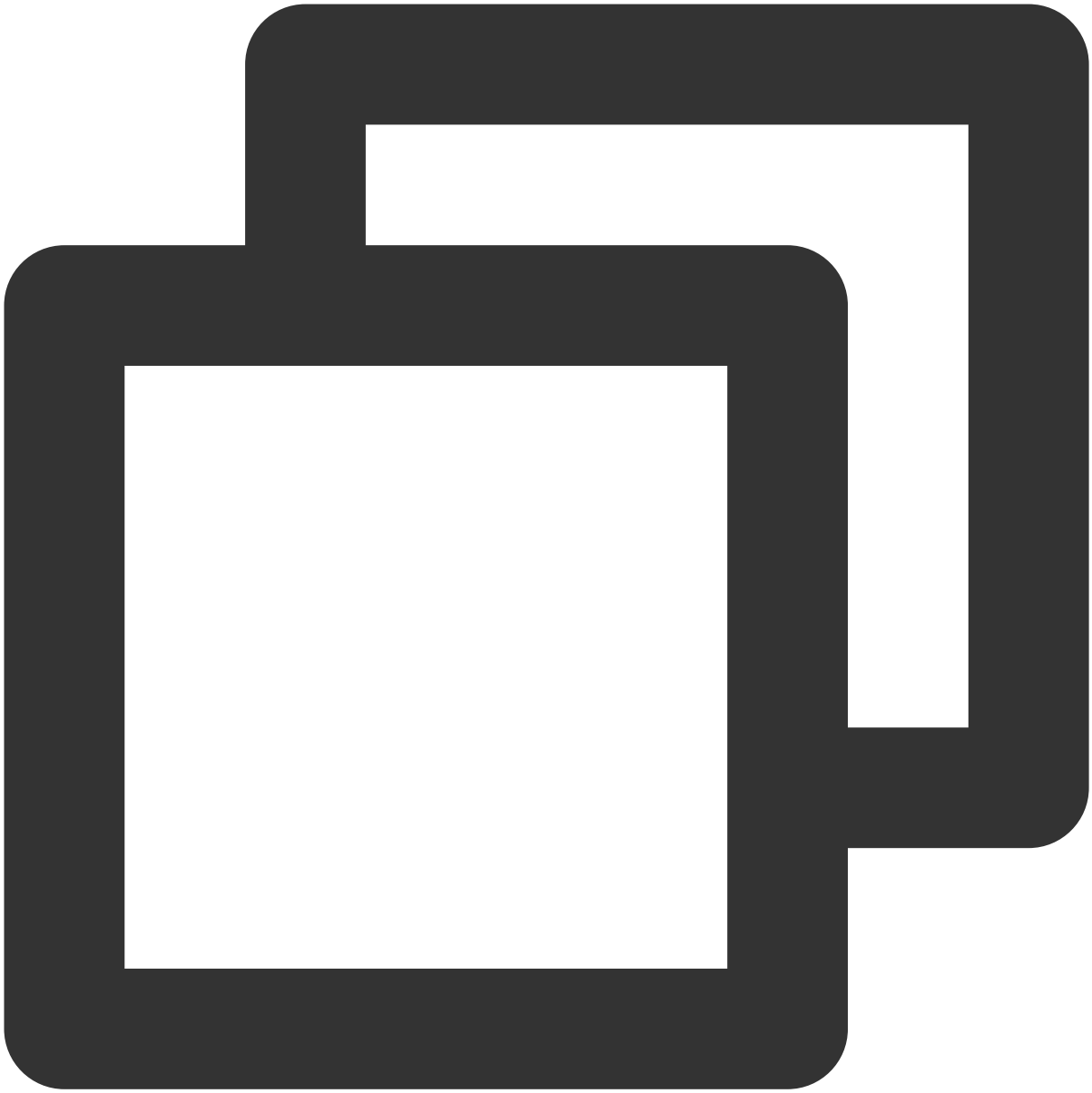
ステップ2：SDK素材リソースパスの設定



```
CGSize previewSize = [self getPreviewSizeByResolution:self.currentPreviewResolution
NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirecto
beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_config
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !isD
    NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfig
    NSError *jsonError;
    NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8StringEncodin
    beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
```

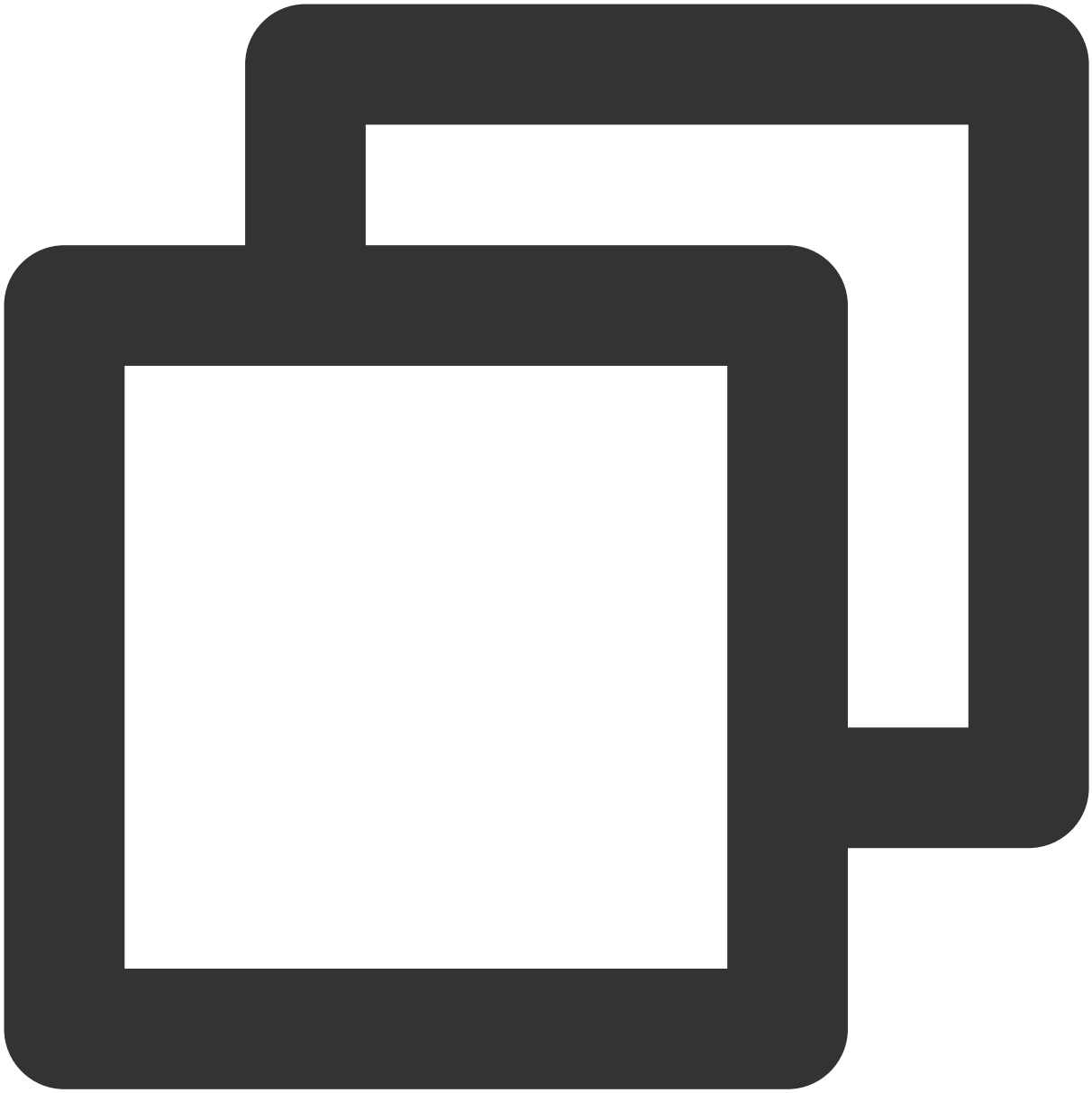
```
                                                options:NSJSONReadingMutableContainers
                                                error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle] bundlePath],
                             @"tnn_"
                             @"beauty_config":beautyConfigJson
};
// Init beauty kit
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDi
```

ステップ3：ログおよびイベント監視の追加



```
 // Register log
[self.beautyKit registerSDKEventListener:self];
[self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL];
```

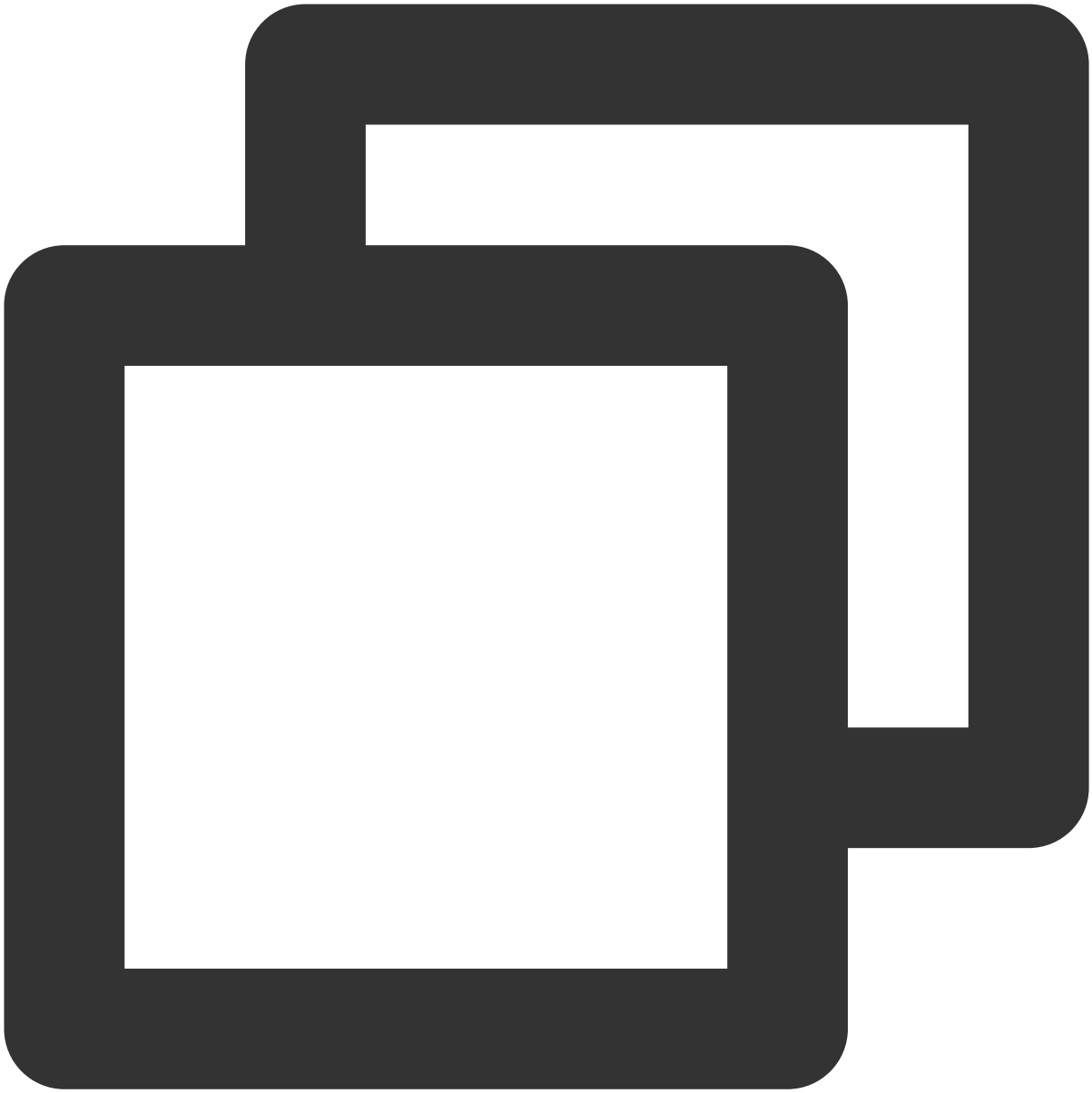
ステップ4：美顔の各種効果の設定



```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString *
```

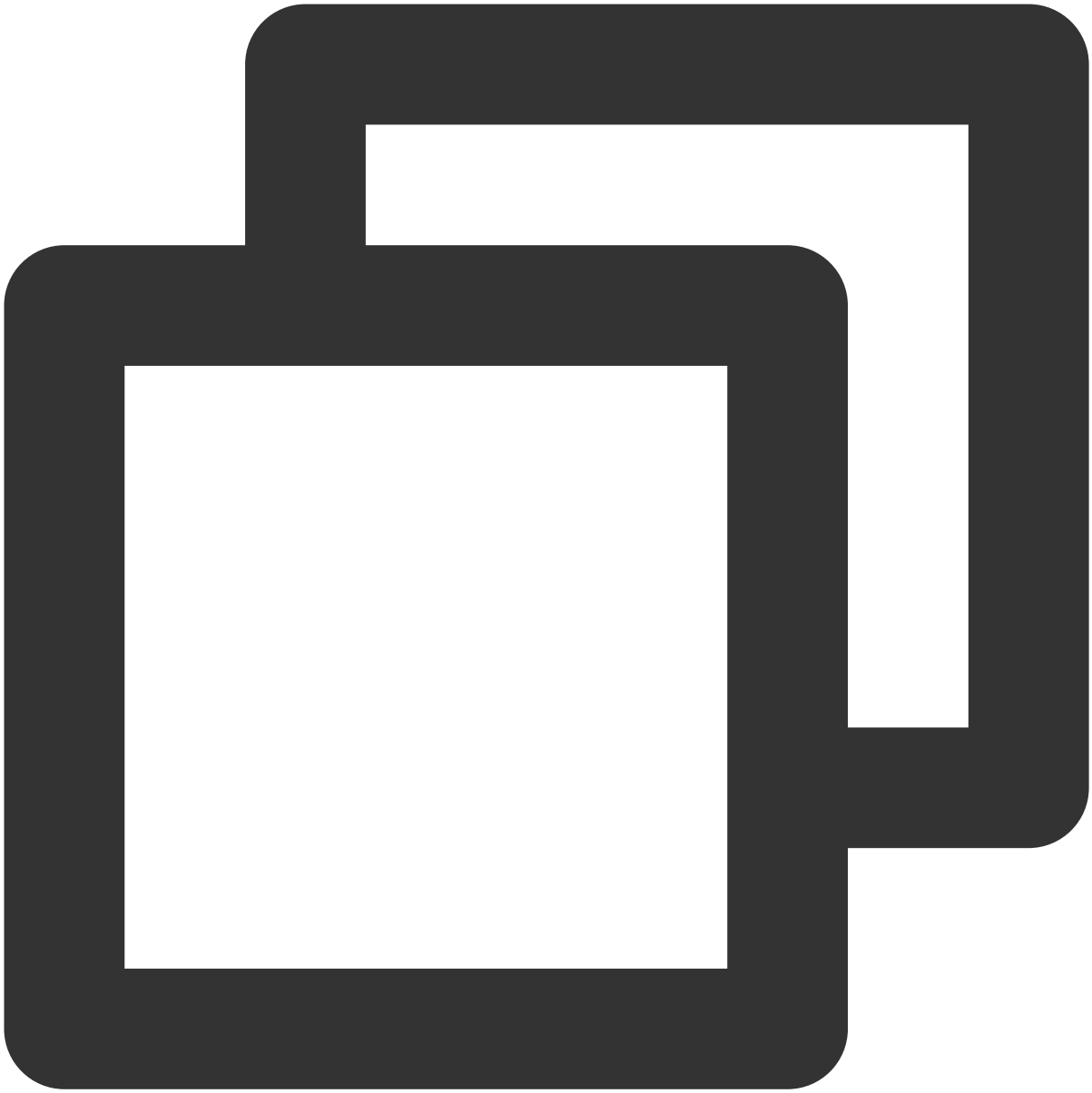
ステップ5：レンダリング処理の実施

ビデオフレームコールバックインターフェースで、YTProcessInputを作成してSDKに渡し、レンダリング処理を行います。DemoのThirdBeautyViewControllerを参照できます。



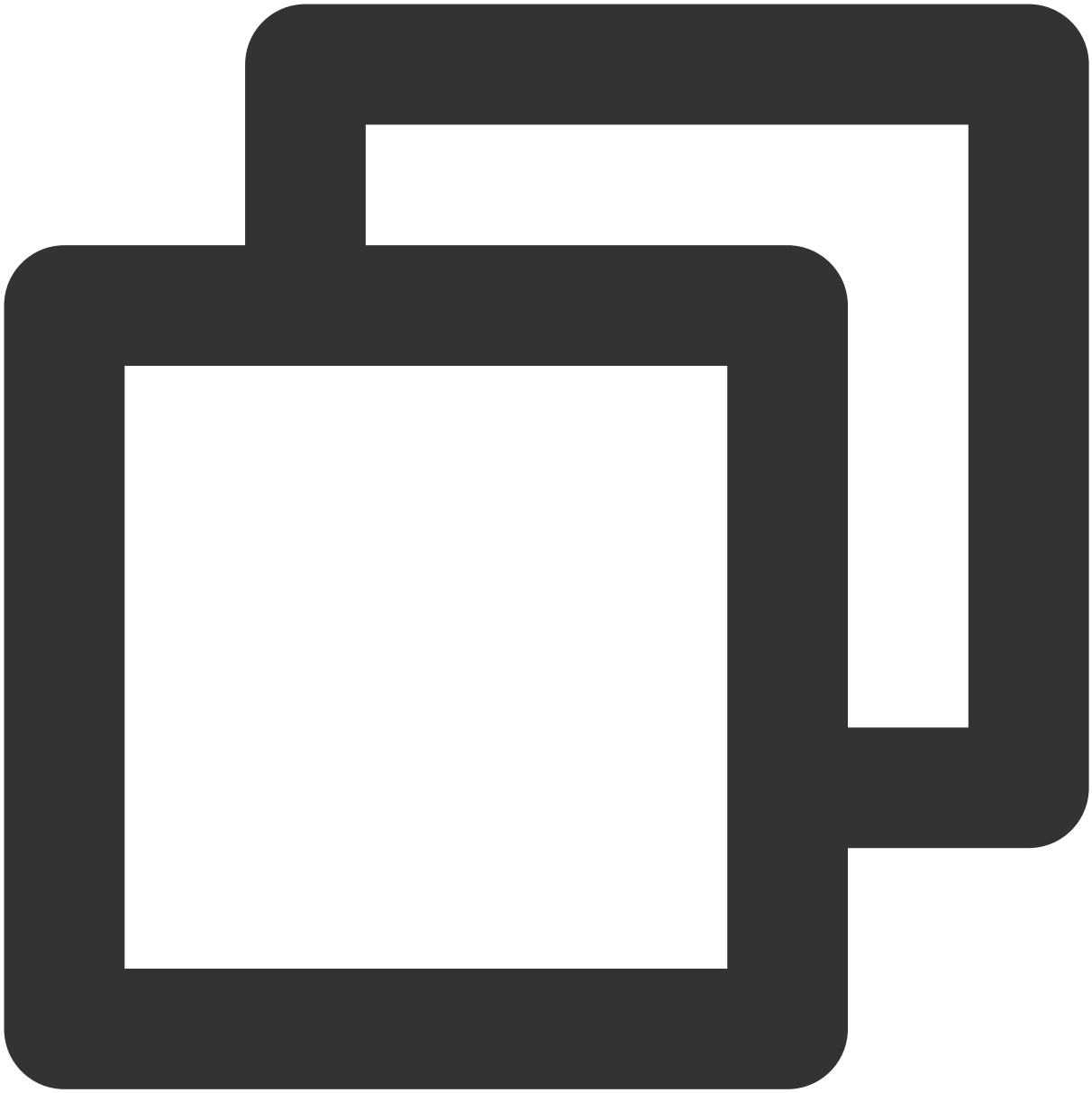
```
[self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft withOrientat
```

ステップ6：SDKの一時停止/再開



```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

ステップ7：レイアウトにSDK美顔パネルを追加



```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
if (@available(iOS 11.0, *)) {
    gSafeInset = [UIApplication sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
    {
        gSafeInset = UIEdgeInsetsZero;
    }
```

```
dispatch_async(dispatch_get_main_queue(), ^{
    //美顔オプションインターフェース
    _vBeauty = [[BeautyView alloc] init];
    [self.view addSubview:_vBeauty];
    [_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
        make.width.mas_equalTo(self.view);
        make.centerX.mas_equalTo(self.view);
        make.height.mas_equalTo(254);
        if(gSafeInset.bottom > 0.0){  // 全画面適用
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
        } else {
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
        }
    }];
    _vBeauty.hidden = YES;
});
```

Android

最終更新日：：2022-07-18 10:06:17

手順1：Demoプロジェクトの解凍

1. Tencent Effect TEを統合したTRTC Demoプロジェクトをダウンロードします。このDemoは、Tencent Effect SDK S1-04パッケージに基づいて作成されています。
2. リソースを置き換えます。このDemoプロジェクトで使用されるSDKパッケージは実際のパッケージと同じではない可能性があるため、このDemoの関連SDKファイルを実際に使用するパッケージのSDKファイルに置き換えてください。具体的な操作は以下のとおりです：
   - xmagicモジュールのlibsディレクトリにある `.aar` ファイルを削除し、SDKのlibsディレクトリにある.aar`ファイルをxmagicモジュールのlibsディレクトリにコピーします。
   - xmagicモジュールのassetsディレクトリにあるすべてのファイルを削除し、SDKの `assets/` ディレクトリにあるすべてのリソースをxmagicモジュールの `../src/main/assets` ディレクトリにコピーします。SDKパッケージのMotionResフォルダにリソースがある場合は、このフォルダを `../src/main/assets` ディレクトリにコピーします。
   - xmagicモジュールのjniLibsディレクトリにあるすべての.soファイルを削除し、SDKパッケージのjniLibsで対応する.soファイルを見つけて（SDKのjinLibsフォルダにあるarm64-v8aおよびarmeabi-v7aの.soファイルが圧縮パッケージに存在しているため、先に解凍する必要がある）、xmagicモジュールの `../src/main/jniLibs` ディレクトリにコピーします。
3. Demoプロジェクトのxmagicモジュールを実際のプロジェクトにインポートします。

手順2：appモジュールのbuild.gradleを開く

1. applicationIdを、テスト用に申請した権限と同じパッケージ名に変更します。
2. gson依存設定を追加します。

```
configurations{
all*.exclude group:'com.google.code.gson'
}
```

手順3：SDKインターフェースの統合

DemoプロジェクトのThirdBeautyActivityクラスを参照できます。

1. 権限承認：

```
//認証に関する注意事項とエラーコードの詳細については、https://www.tencentcloud.com/document/product/1143/45385#step-1.-authenticateを参照してください。
XMagicImpl.checkAuth((errorCode, msg) -> {
if (errorCode == TELicenseCheck.ERROR_OK) {
showLoadResourceView();
}else{
TXCLog.e(TAG, "認証に失敗しました。認証urlおよびkeyをご確認ください" + errorCode + " "
+ msg);
}
});
```

2. 素材の初期化：

```
private void showLoadResourceView() {
if (XmagicLoadAssetsView.isCopyedRes) {
XmagicResParser.parseRes(getApplicationContext());
initXMagic();
}else{
loadAssetsView = new XmagicLoadAssetsView(this);
loadAssetsView.setOnAssetsLoadFinishListener(() -> {
XmagicResParser.parseRes(getApplicationContext());
initXMagic();
});
}
}
```

3. プッシュ設定の有効化：

```
mTRTCCloud.setLocalVideoProcessListener(TRTCCloudDef.TRTC_VIDEO_PIXEL_FORMAT_Texture_2D, TRTCCloudDef.TRTC_VIDEO_BUFFER_TYPE_TEXTURE, new TRTCCloudListener.TRTCVideoFrameListener() {
@Override
public void onGLContextCreated() {
}
@Override
public int onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame, TRTCCloudDef.TRTCVideoFrame dstFrame) {
}
@Override
public void onGLContextDestory() {
```

```
}
});
```

4. **textureIdをSDKに渡し、レンダリング処理を実施：**
TRTCVideoFrameListenerインターフェースの `onProcessVideoFrame(TRTCCloudDef.TRTCVideoFrame srcFrame, TRTCCloudDef.TRTCVideoFrame dstFrame)` メソッド内に次のコードを追加します：

```
dstFrame.texture.textureId = mXMagic.process(srcFrame.texture.textureId, srcFrame.width, srcFrame.height);
```

5. **SDKの一時停止/停止：**
onPause()は美顔効果の一時停止に使用し、Activity/Fragmentライフサイクルメソッドにおいて実行できます。onDestroyメソッドはGLスレッドで呼び出してください（onTextureDestroyedメソッドでXMagicImplオブジェクトの `onDestroy()` を呼び出すことができます）。その他の使用についてはDemoをご参照ください。

```
mXMagic.onPause(); //一時停止。ActivityのonPauseメソッドにバインドします
mXMagic.onDestroy(); //破棄。GLスレッドで呼び出してください
```

6. **レイアウトにSDK美顔パネルを追加：**

```
<RelativeLayout
android:layout_above="@+id/ll_edit_info"
android:id="@+id/livepusher_bp_beauty_pannel"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
```

7. **パネルの初期化：**

```
private void initXMagic() {
if (mXMagic == null) {
mXMagic = new XMagicImpl(this, mBeautyPanelView);
}else{
mXMagic.onResume();
}
}
```

具体的な操作についてはDemoプロジェクトの `ThirdBeautyActivity.initXMagic();` メソッドをご参照ください。

Flutter

最終更新日：：2023-04-27 16:15:18

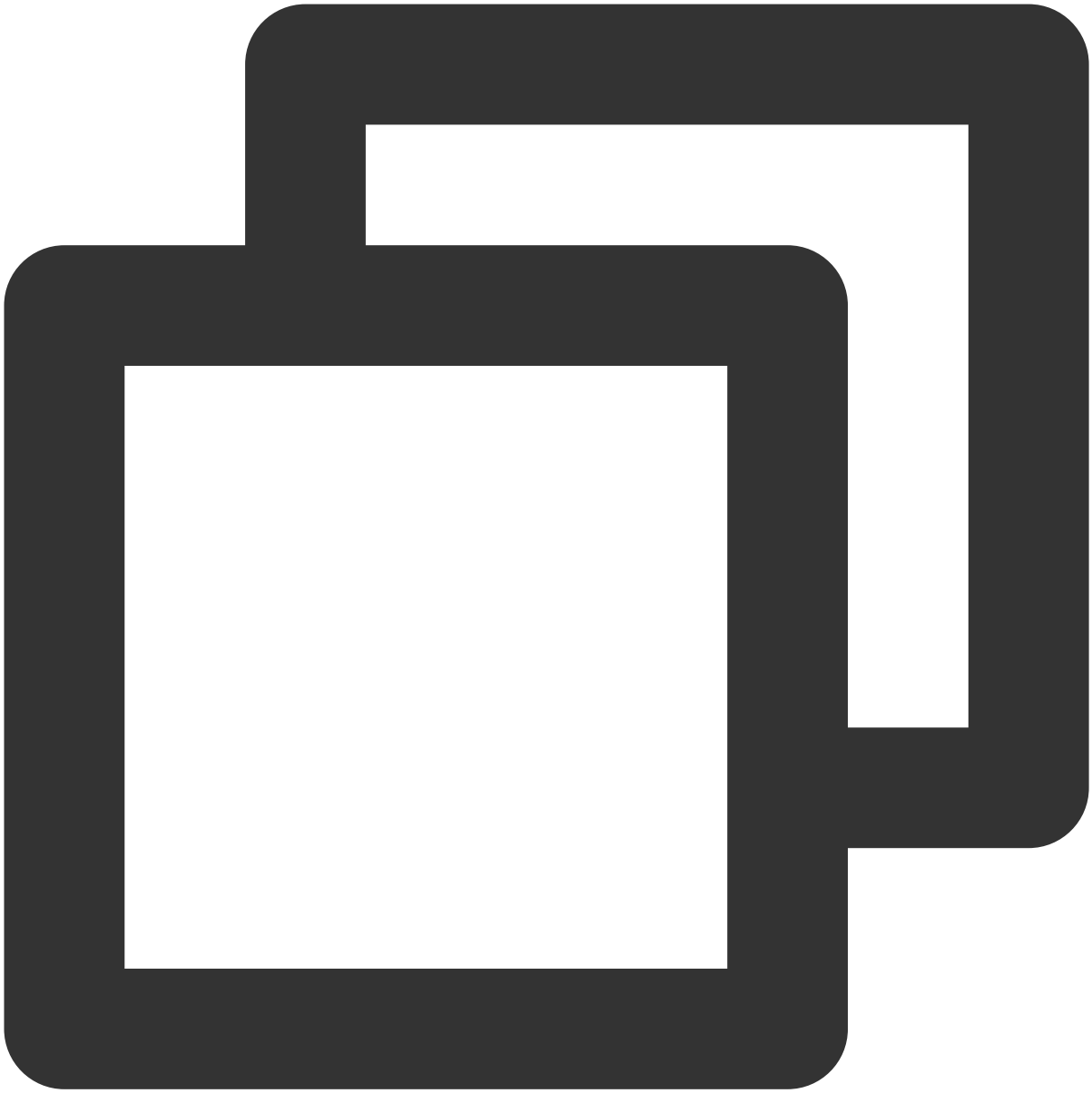
ステップ1：美顔リソースのダウンロードと統合

1. 購入したパッケージに応じてSDKのダウンロードを行います。
2. ファイルをご自身のプロジェクトに追加します。

Android

iOS

1. appモジュール下でbuild.gradleファイルを見つけ、対応するパッケージのmaven参照アドレスを追加します。例えばS1-04パッケージを選択した場合は下記を追加します。

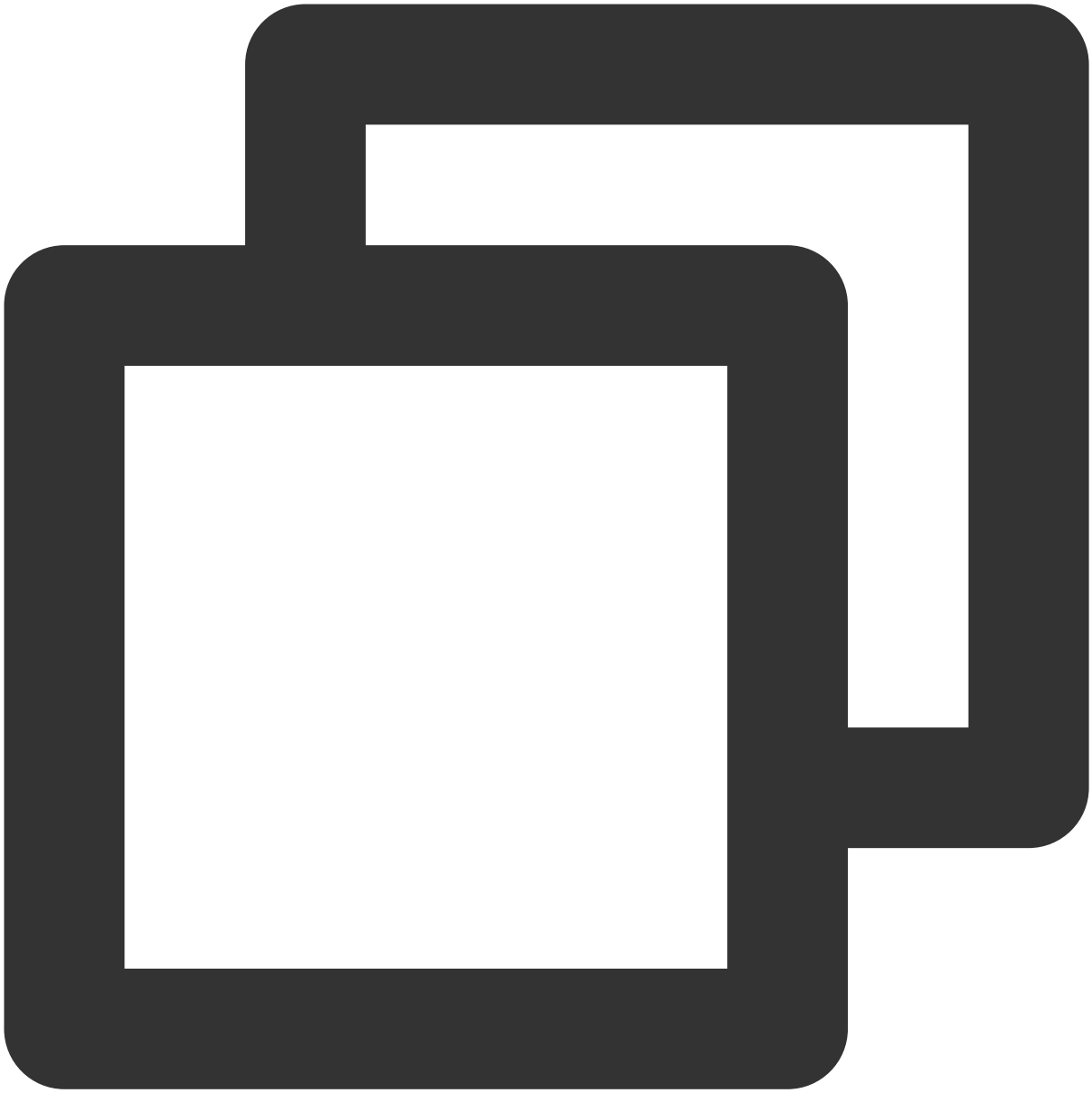


```
dependencies{
    implementation 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
}
```

各パッケージに対応するmavenアドレスについては、ドキュメントをご参照ください。

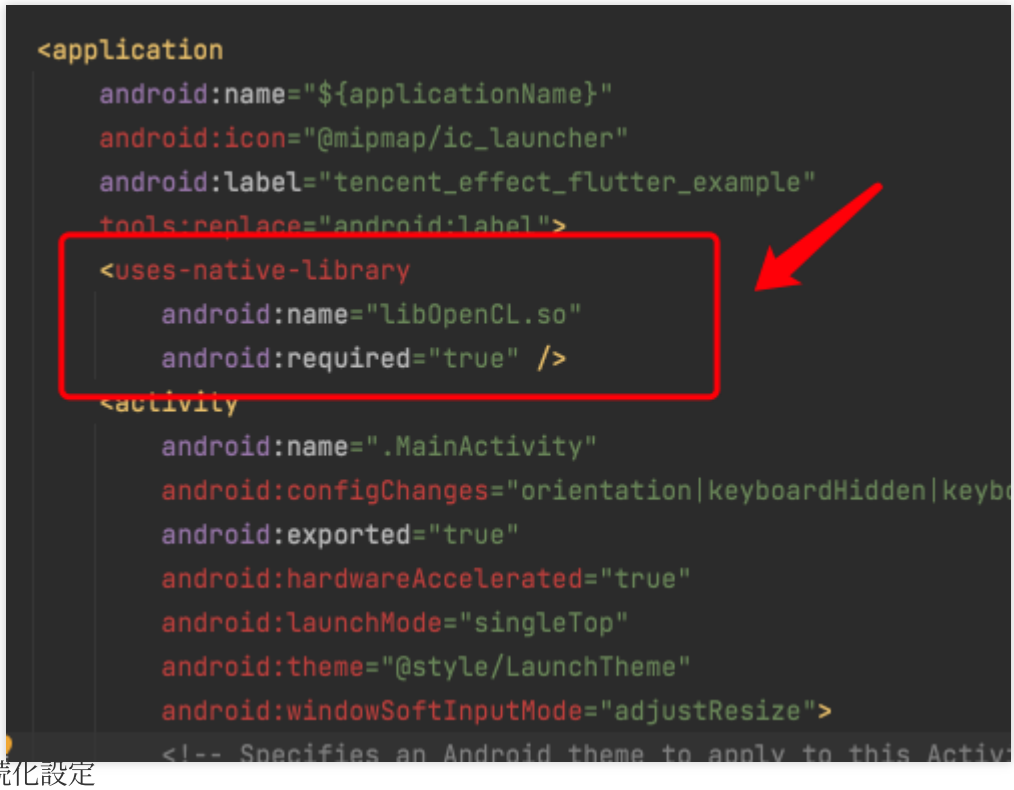
2. appモジュール下でsrc/main/assetsフォルダを見つけます。存在しない場合は作成し、ダウンロードしたSDKパッケージにMotionResフォルダがあるかどうかをチェックし、もしあればそのフォルダを `../src/main/assets` ディレクトリ下にコピーします。

3. appモジュール下でAndroidManifest.xmlファイルを見つけ、applicationテーブルに次のタグを追加します



```
<uses-native-library
       android:name="libOpenCL.so"
       android:required="true" />
       //ここでtrueは、このライブラリがないとアプリが正常に動作しないことを意味します。システムは
       //falseは、アプリケーションがこのライブラリ （存在する場合）を使用できますが、特に（必要
       //Android公式サイトの説明：%!s(<nil>)
```

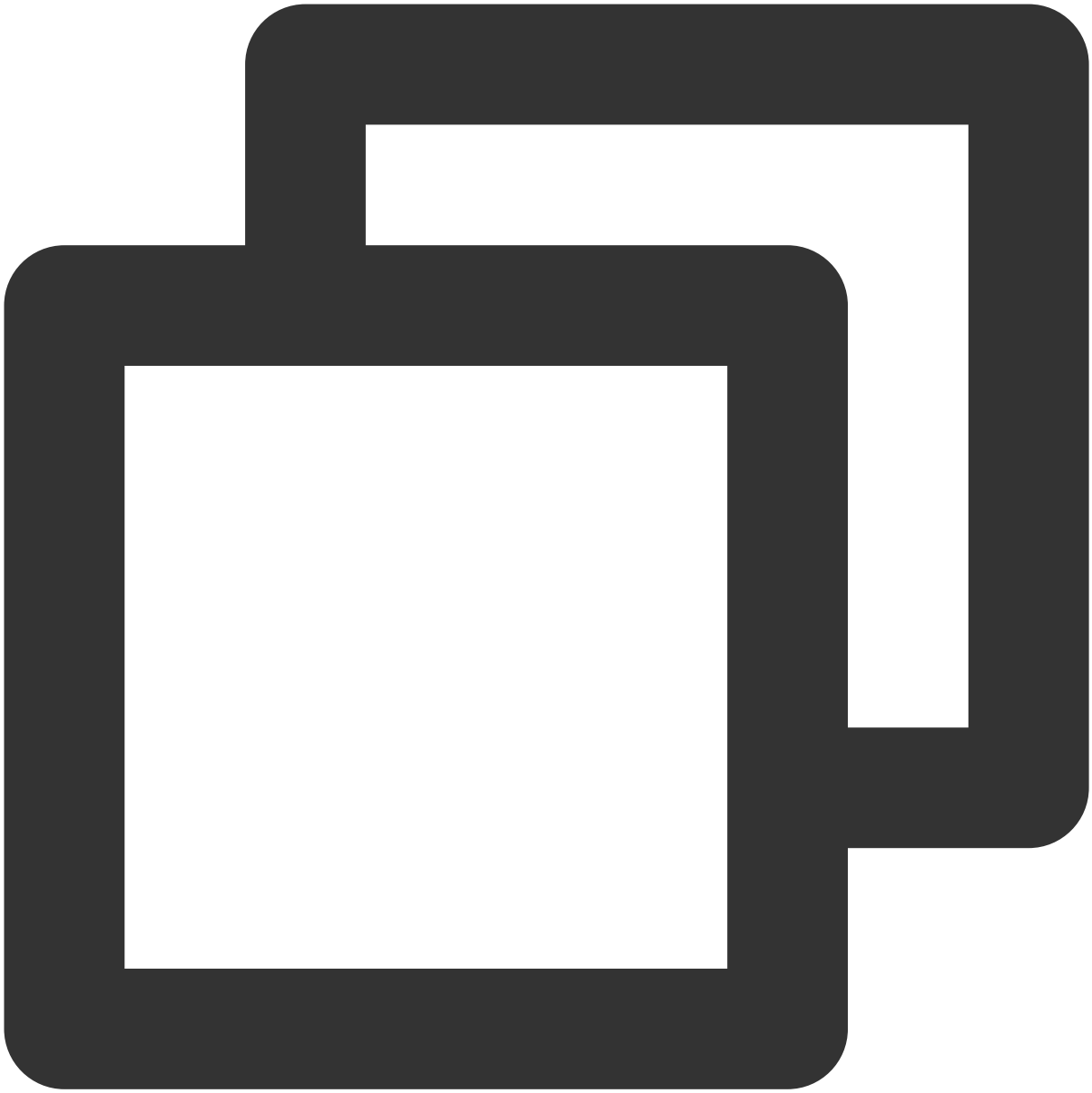
追加すると下図のようになります。



4. 難読化設定

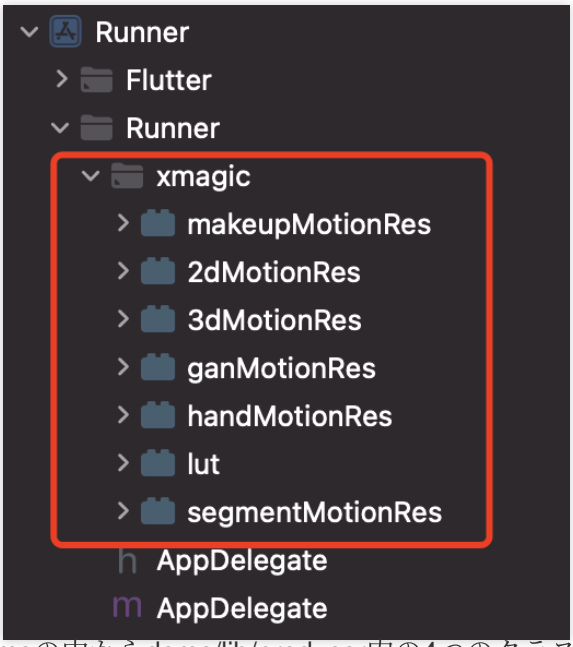
releaseパッケージを作成する際、コンパイルの最適化を有効（minifyEnabledをtrueに設定）にしていると、javaレイヤーで呼び出されないコードがカットされる場合があります。これらのコードはnativeレイヤーで呼び出される場合があり、 `no xxx method` の異常が生じることがあります。

このようなコンパイル最適化を有効にしている場合は、これらのkeepルールを追加し、xmagicのコードがカットされないようにする必要があります。



```
-keep class com.tencent.xmagic.** { *;}
-keep class org.light.** { *;}
-keep class org.libpag.** { *;}
-keep class org.extra.** { *;}
-keep class com.gyailib.**{ *;}
-keep class com.tencent.cloud.iai.lib.** { *;}
-keep class com.tencent.beacon.** { *;}
-keep class com.tencent.qimei.** { *;}
```

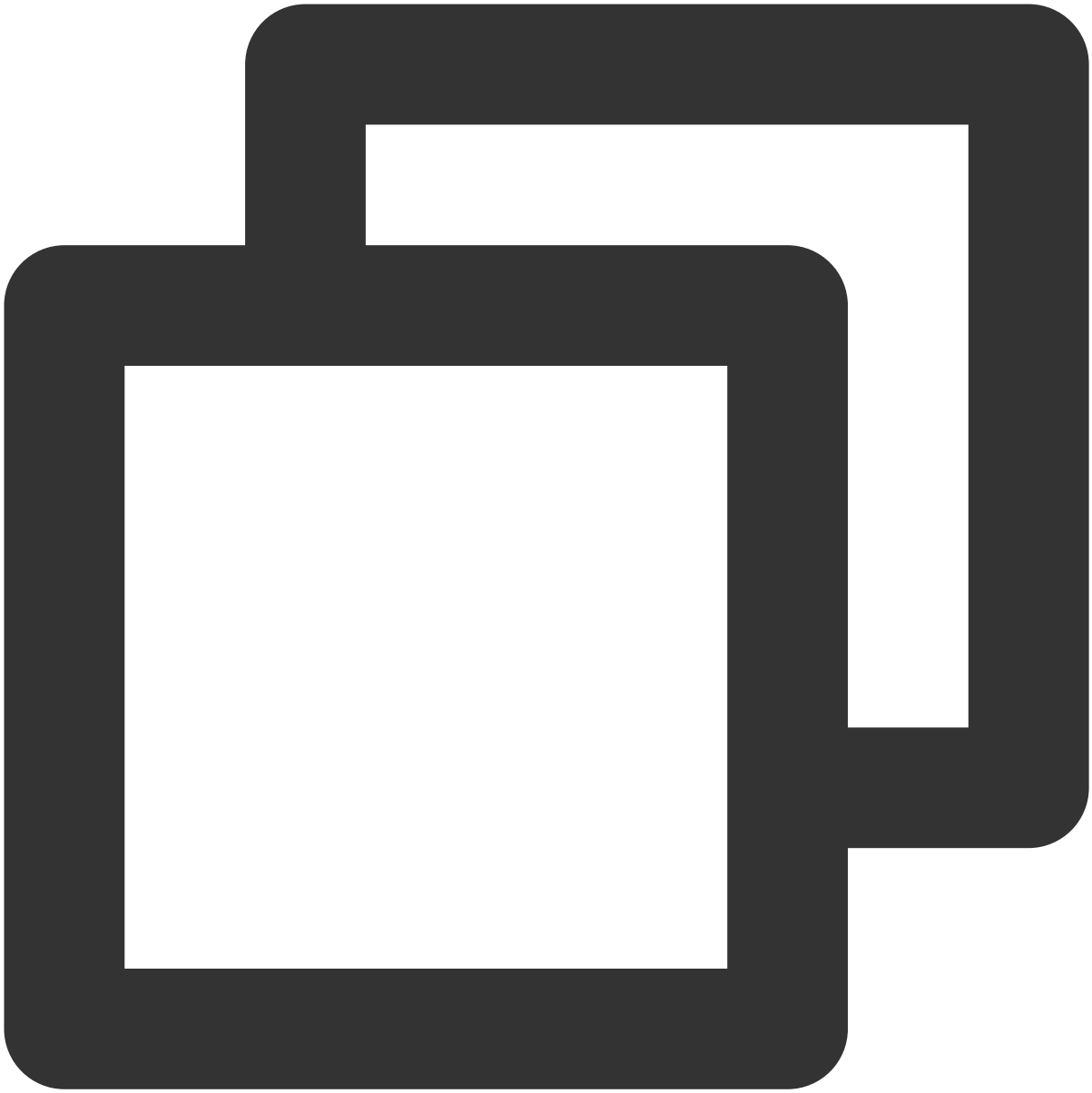
1. 美顔リソースをプロジェクトに追加します。追加すると下図のようになります（リソースの種類と下図は完全には一致しません）。



2. ｄemoの中からdemo/lib/producer内の4つのクラスである、BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid、BeautyPropertyProducerIOSをコピーしてご自身のFlutterプロジェクトに追加します。これらの4つのクラスは美顔リソースを設定し、美顔タイプを美顔パネルに表示するために用いられます。

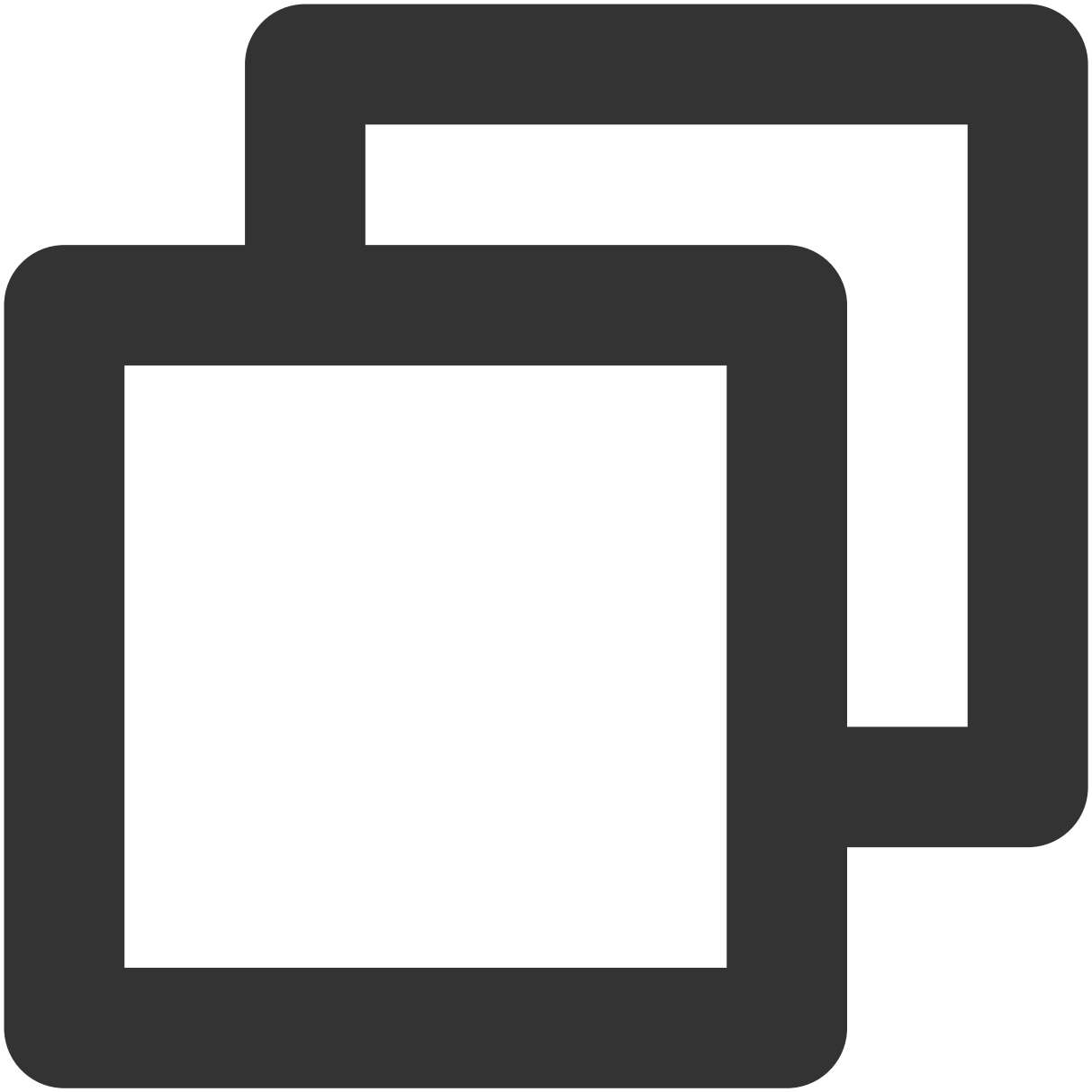
ステップ2：FlutterバージョンSDKの参照

GitHub参照：プロジェクトのpubspec.yamlファイルに下記の参照を追加します。



```
tencent_effect_flutter:
  git:
    url: https://github.com/TencentCloud/tencenteffect-sdk-flutter
```

ローカル参照：tencent_effect_flutterから、最新バージョンのtencent_effect_flutterをダウンロードし、フォルダ `android` 、 `ios` 、 `lib` とファイル `pubspec.yaml` 、 `tencent_effect_flutter.iml` をプロジェクトディレクトリに追加します。そのあと、プロジェクトのpubspec.yamlファイルの中に以下の参照を追加します。（参考用demo）



```
tencent_effect_flutter:
    path: ../
```

tencent_effect_flutterは、1つのブリッジングヘッダーを提供するのみです。その中で依存するXMagicはデフォルトで最新バージョンになっています。実際に美顔を実現するのはXMagicです。

最新バージョンの美顔SDKを利用される場合は、以下のステップでSDKのアップグレードが行えます。

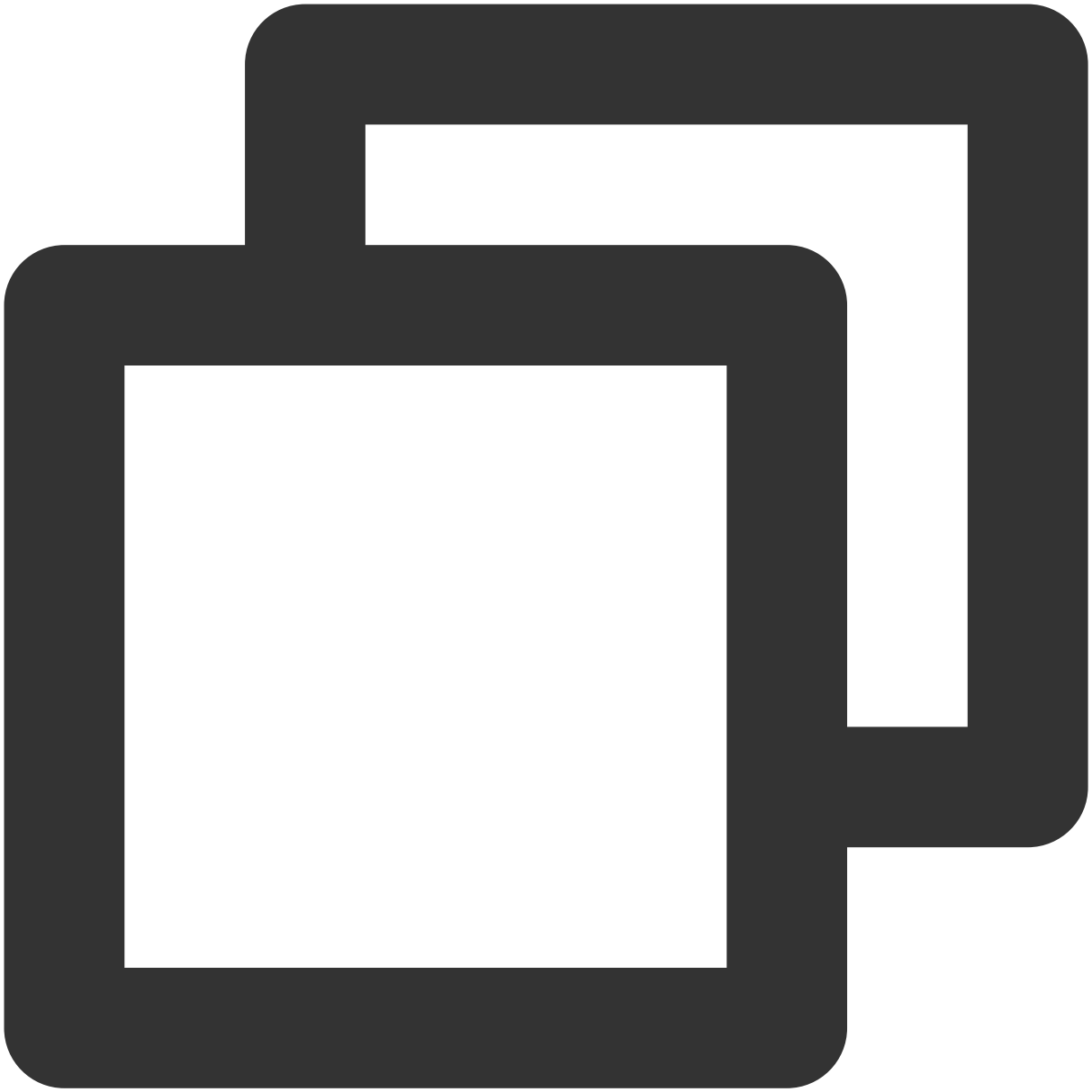
Android

iOS

プロジェクトディレクトリでコマンド： `flutter pub upgrade` を実行、あるいは `subspec.yaml` ページの右上隅の**Pub upgrade**をクリックします。
プロジェクトディレクトリでコマンド：flutter pub upgradeを実行、その後、iOSディレクトリのコマンド： `pod update` を実行します。

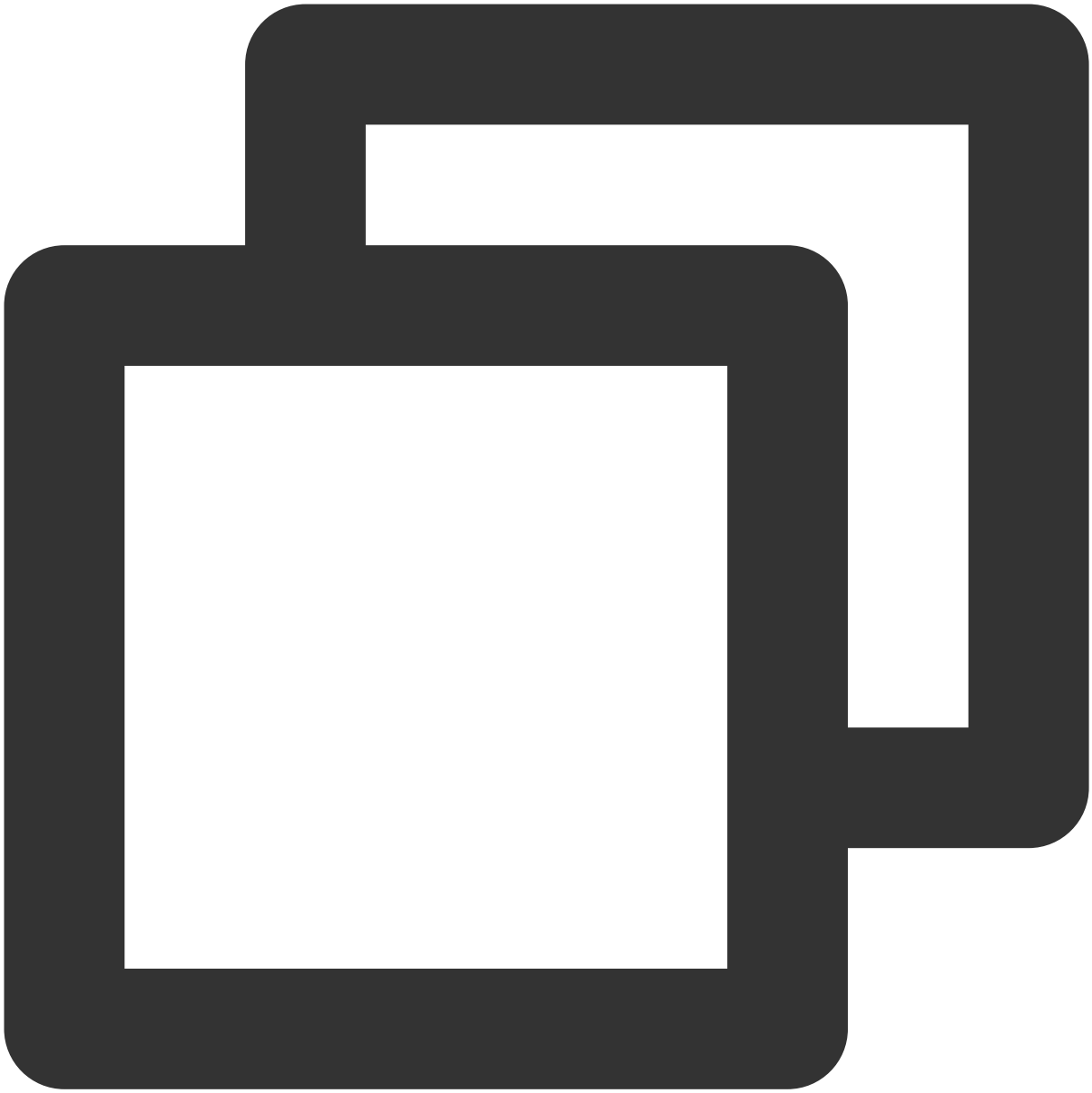
ステップ3：TRTCとのバインド

Android
iOS
アプリケーションのapplicationクラスのoncreateメソッド（またはFlutterActivityのonCreateメソッド）に次のコードを追加します。



```
TRTCCloudPlugin.register(new XmagicProcesserFactory());
```

アプリケーションのAppDelegateクラスのdidFinishLaunchingWithOptionsメソッドに次のコードを追加します。

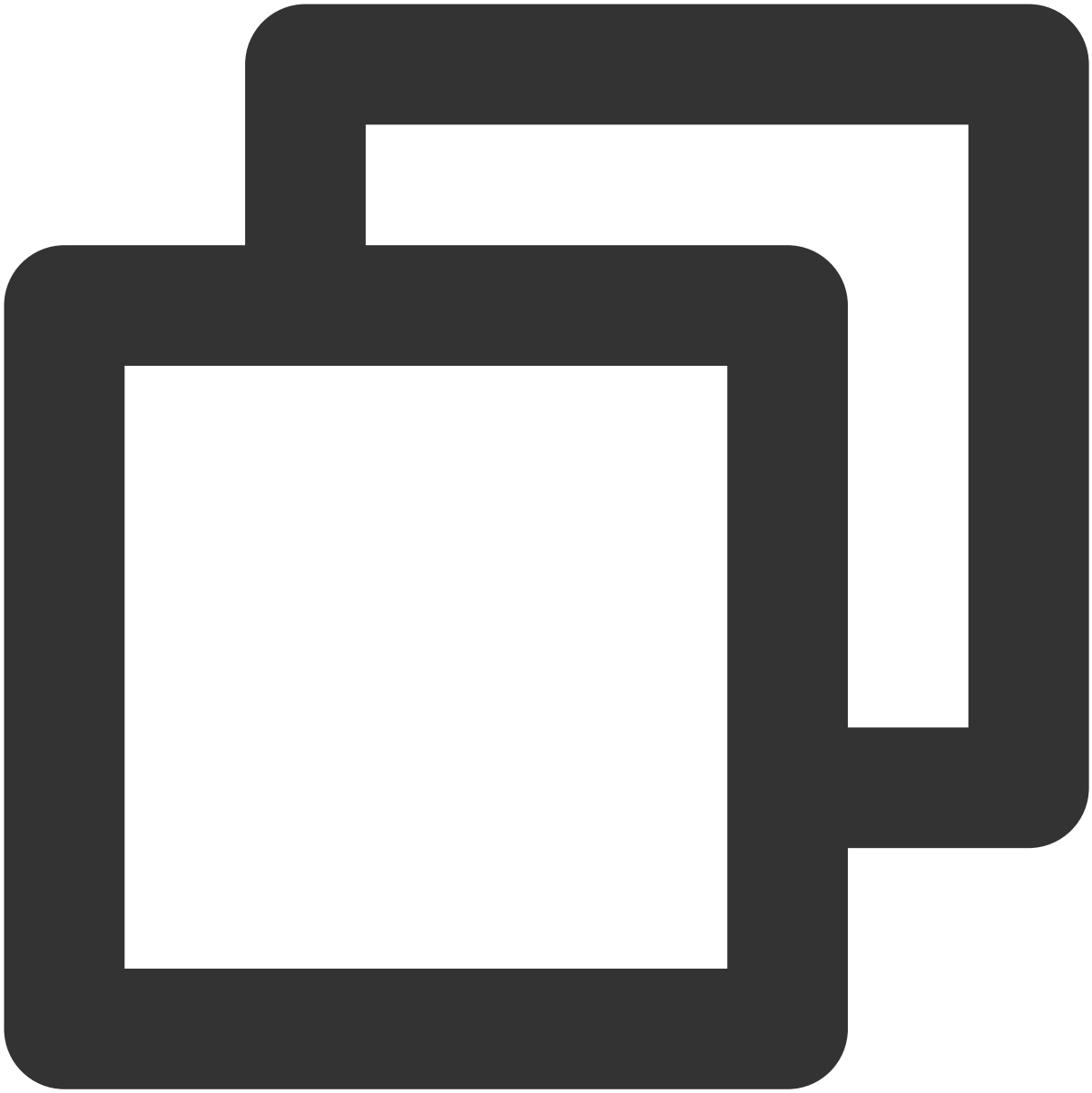


```
XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc] init];
[TencentTRTCCloud registerWithCustomBeautyProcesserFactory:instance];
```

追加すると下図のようになります。

```objc
#import "AppDelegate.h"
#import "GeneratedPluginRegistrant.h"
@import live_flutter_plugin;
@import tencent_effect_flutter;
@import tencent_trtc_cloud;

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  [GeneratedPluginRegistrant registerWithRegistry:self];
  // Override point for customization after application launch.
  XmagicProcesserFactory *instance = [[XmagicProcesserFactory alloc
  [TXLivePluginManager registerWithCustomBeautyProcesserFactory:inst
  [TencentTRTCCloud registerWithCustomBeautyProcesserFactory:instan
  return [super application:application didFinishLaunchingWithOptio
}

@end
```

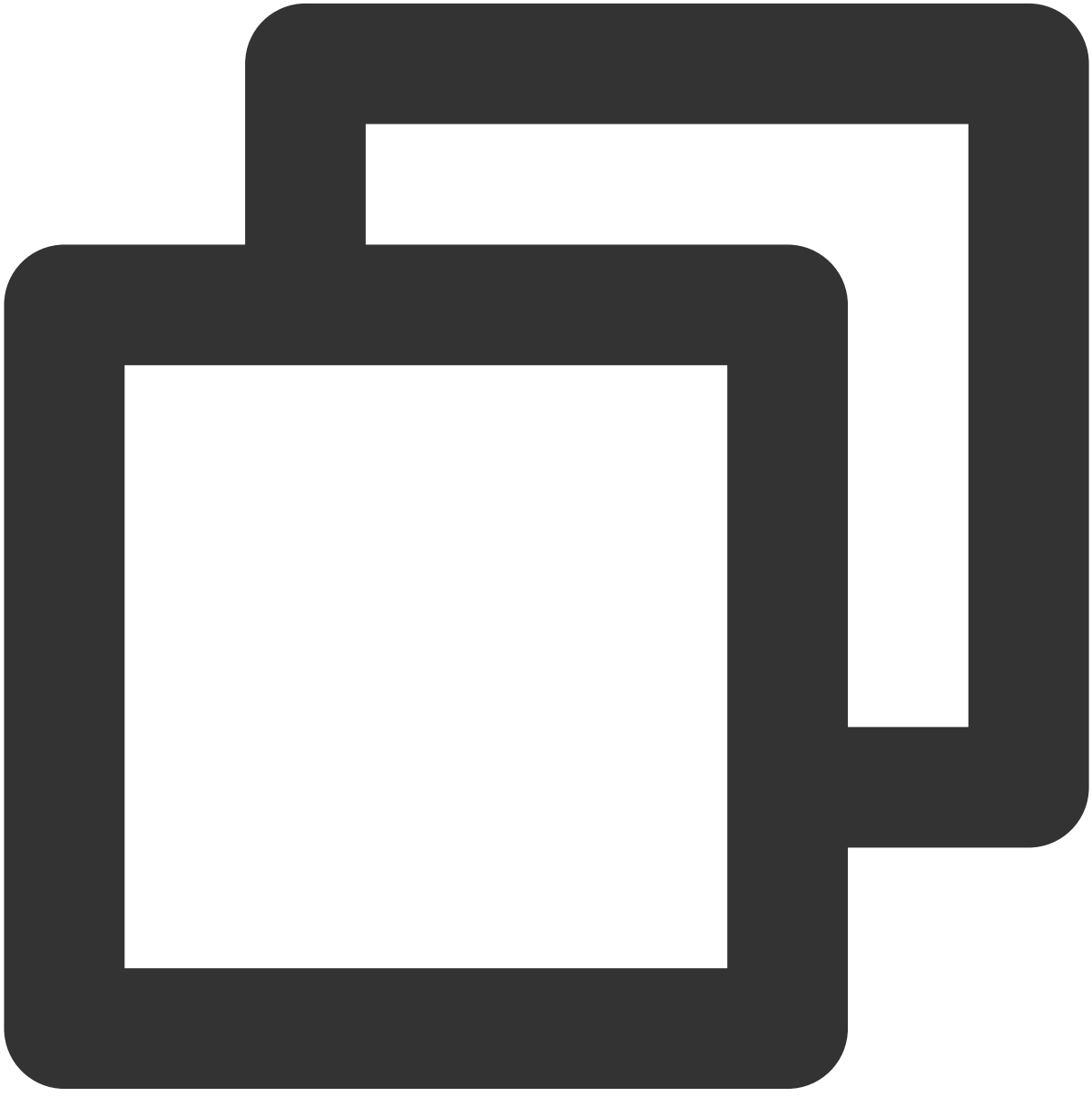
ステップ4：リソース初期化インターフェースの呼び出し



```
String dir =  await BeautyDataManager.getInstance().getResDir();
 TXLog.printlog('ファイルパス：$dir');
 TencentEffectApi.getApi()?.initXmagic(dir,(reslut) {
   _isInitResource = reslut;
   callBack.call(reslut);
   if (!reslut) {
     Fluttertoast.showToast(msg: "リソースの初期化に失敗しました");
   }
 }); TencentEffectApi.getApi()?.initXmagic((reslut) {
   if (!reslut) {
     Fluttertoast.showToast(msg: "リソースの初期化に失敗しました");
```

```
        }
    });
```

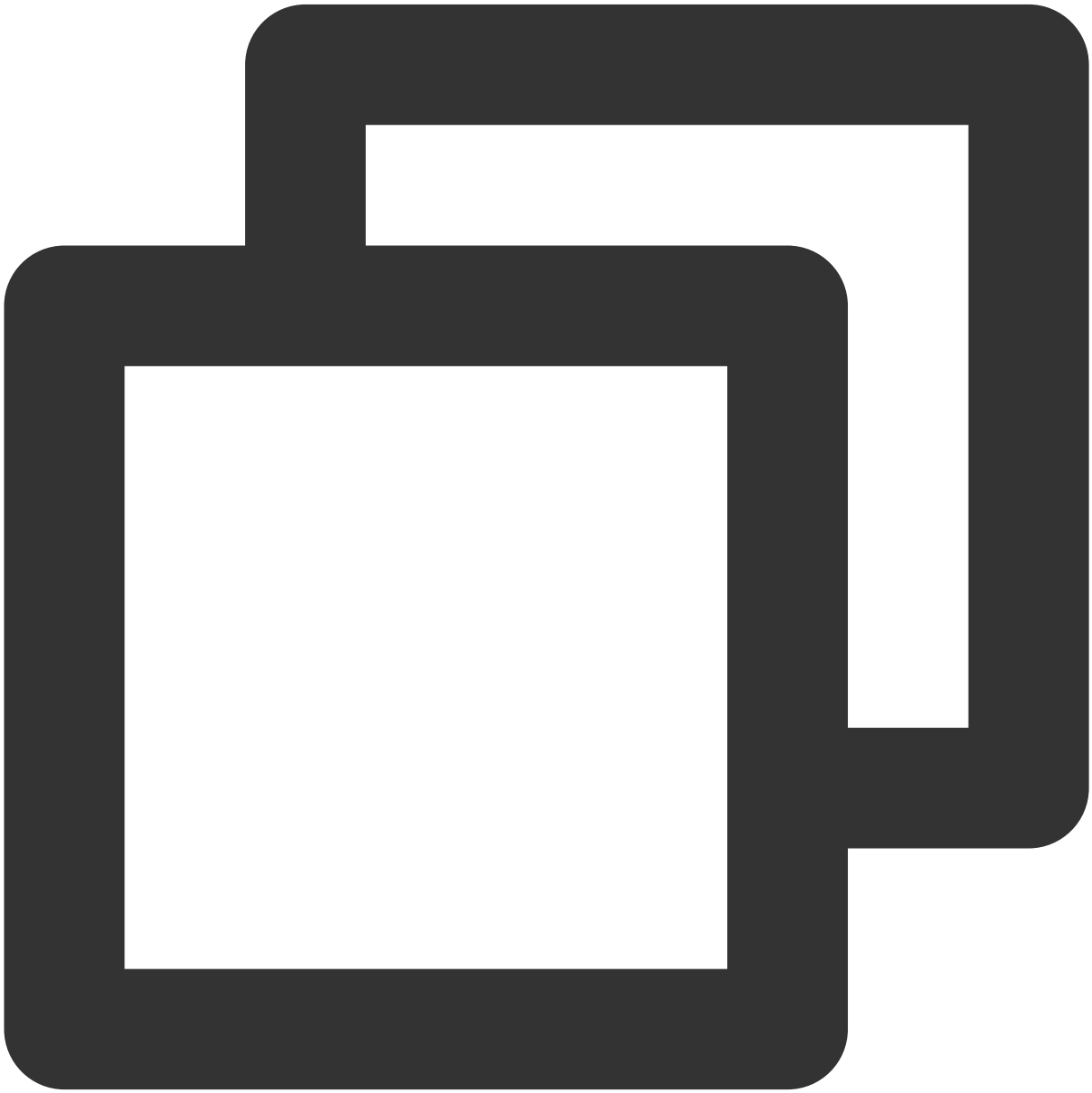
ステップ5：美顔権限承認を実行



```
TencentEffectApi.getApi()?.setLicense(licenseKey, licenseUrl,
            (errorCode, msg) {
          TXLog.printlog("認証結果を印刷します errorCode = $errorCode   msg = $msg");
```

```
        if (errorCode == 0) {
            //認証に成功しました
        }
    });
```

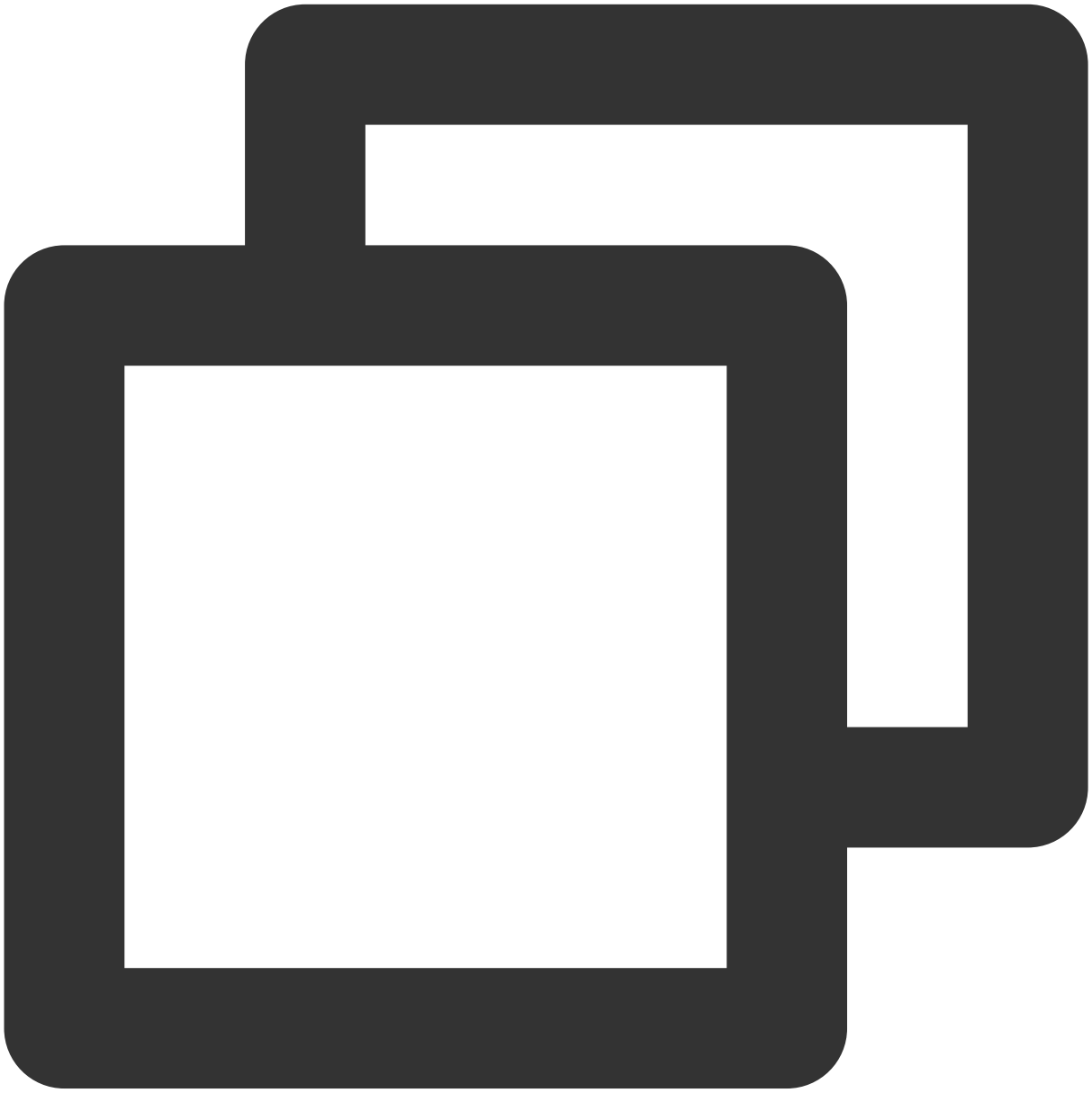
ステップ6：美顔を有効にする



```
///美顔操作を有効にします
```

```
var enableCustomVideo = await trtcCloud.enableCustomVideoProcess(open);
```

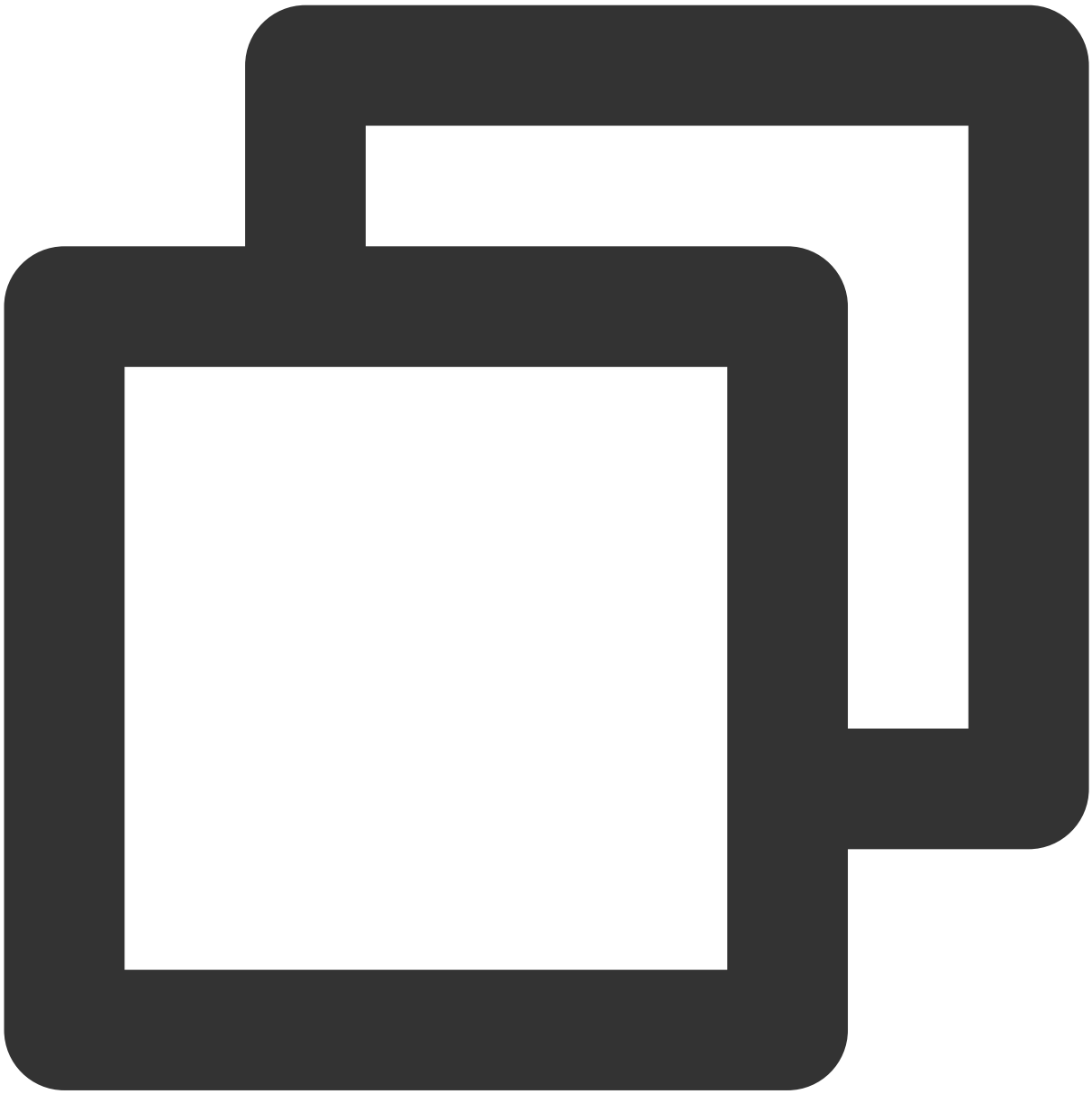
ステップ7：美顔属性の設定



```
TencentEffectApi.getApi()?.updateProperty(_xmagicProperty!);
///_xmagicPropertyではBeautyDataManager.getInstance().getAllPannelData();によってすべ
```

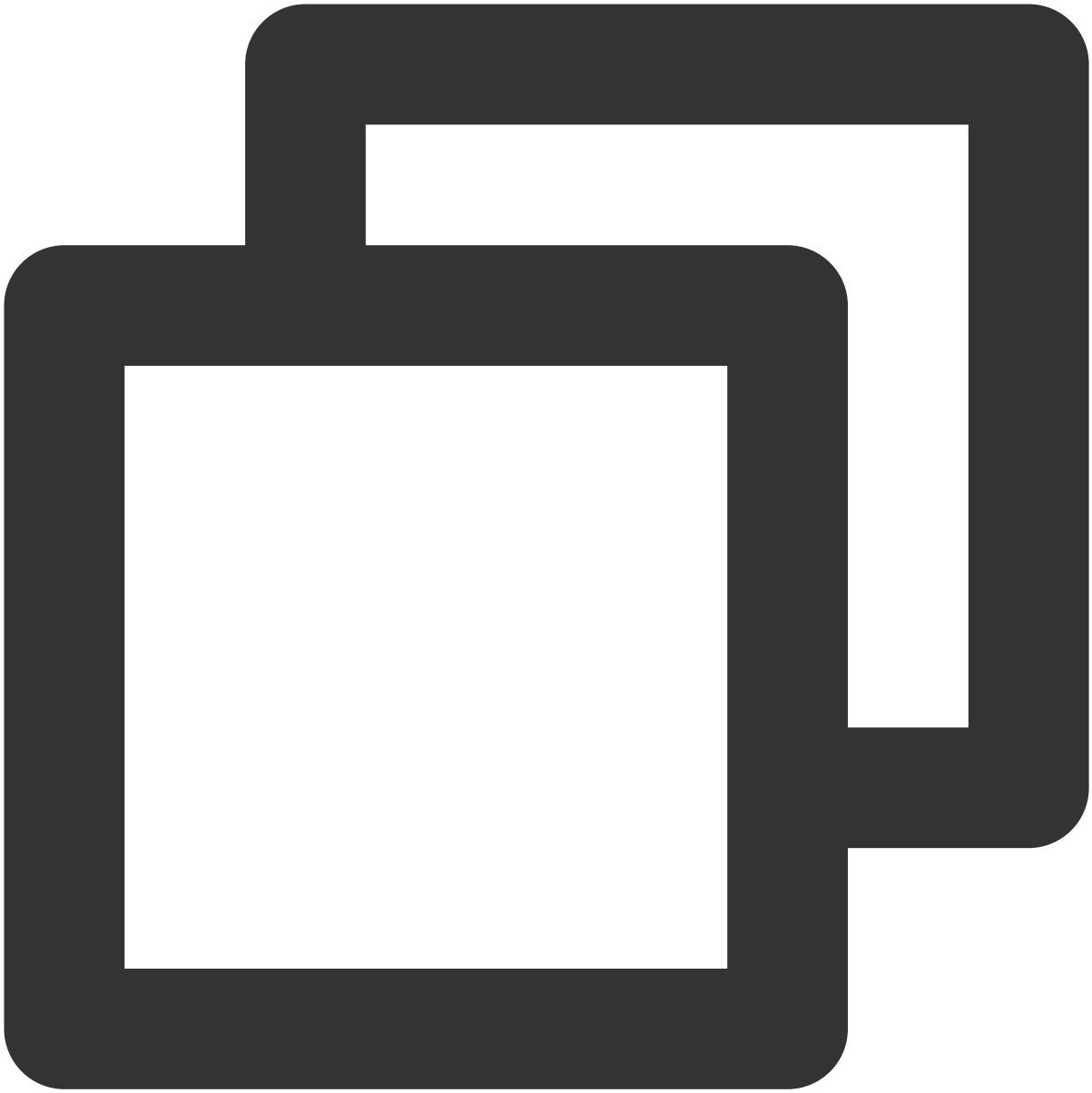
ステップ8：その他の属性の設定

美顔サウンドエフェクトの一時停止



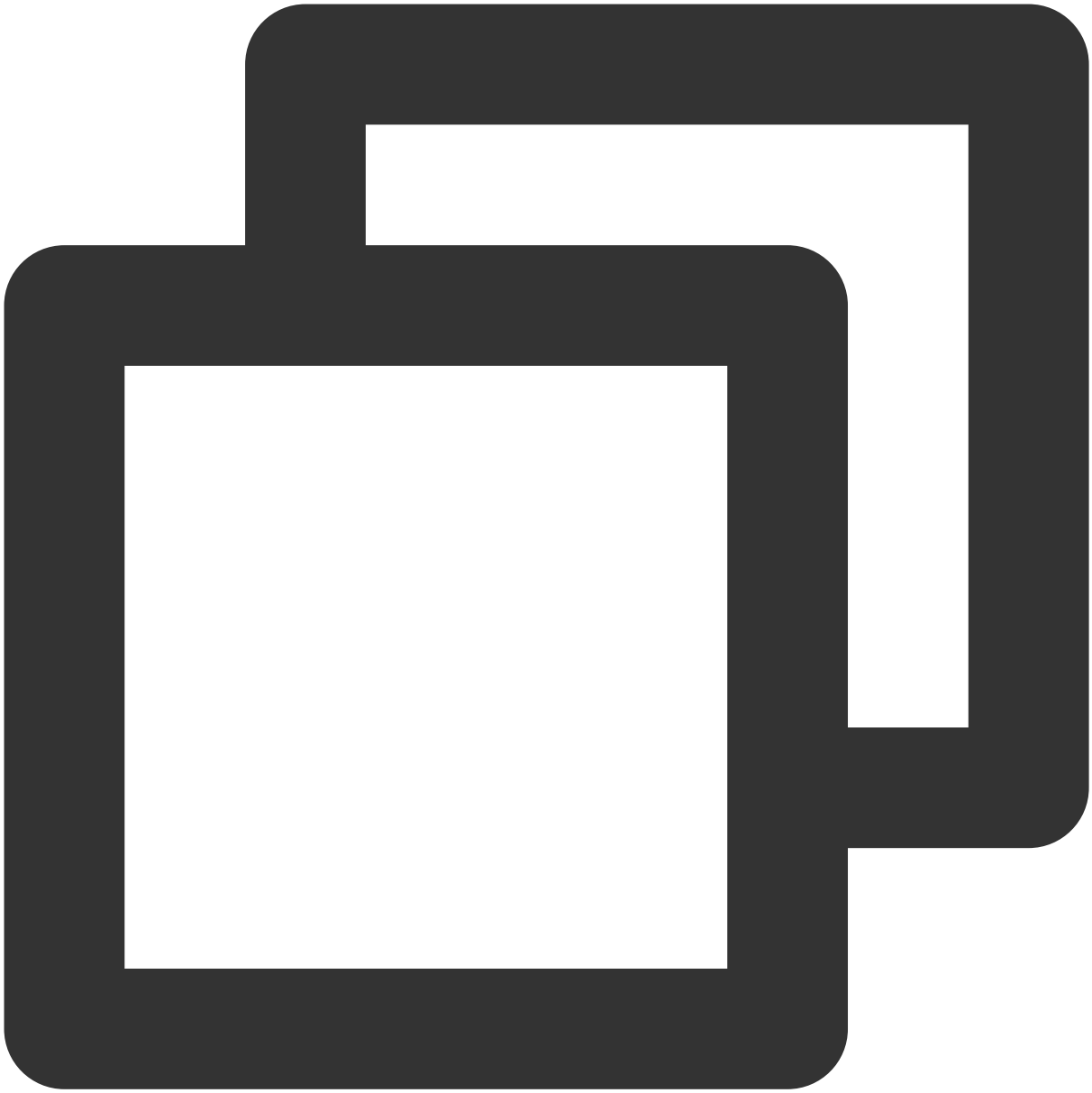
```
TencentEffectApi.getApi()?.onPause();
```

美顔サウンドエフェクトの再開



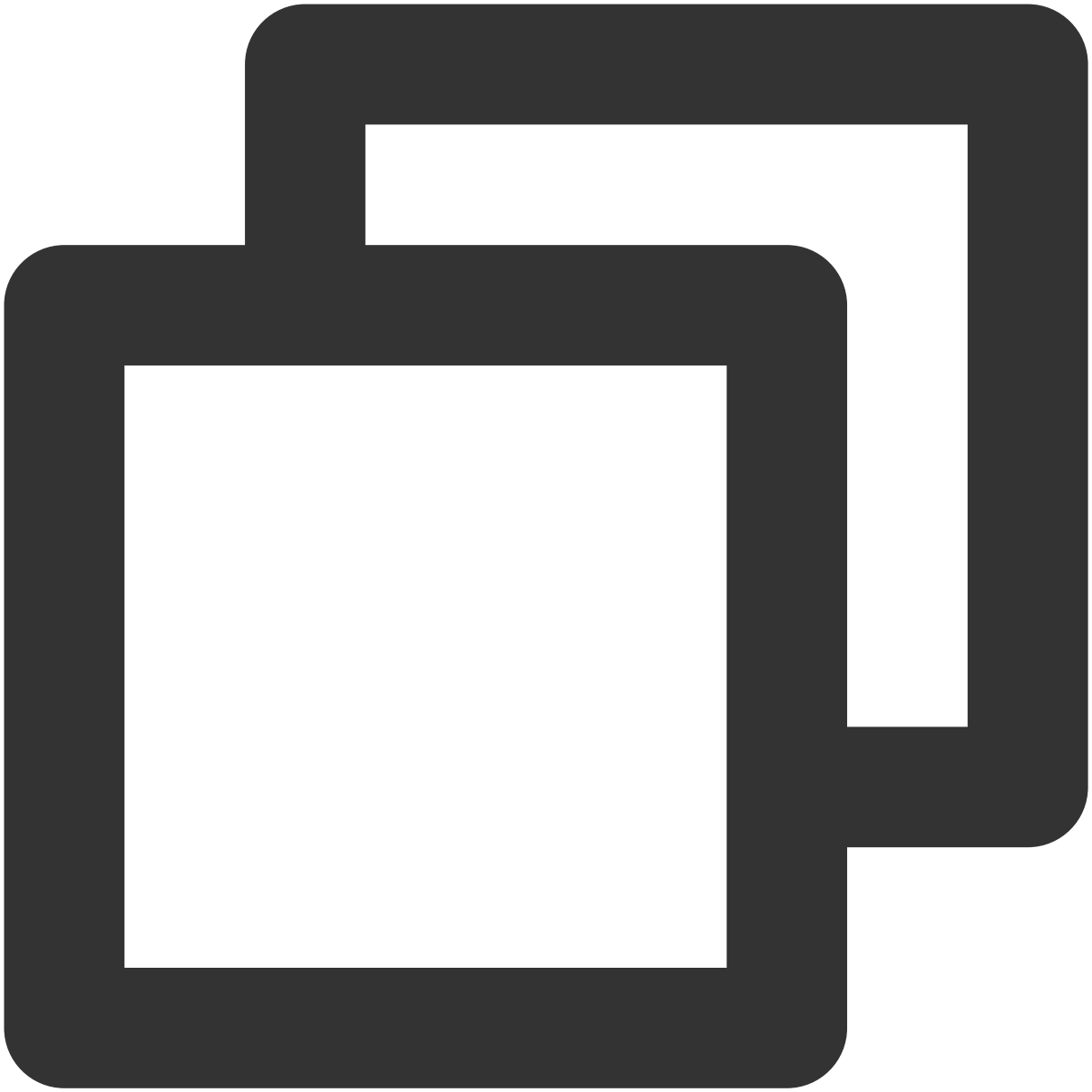
```
TencentEffectApi.getApi()?.onResume();
```

美顔イベントの監視



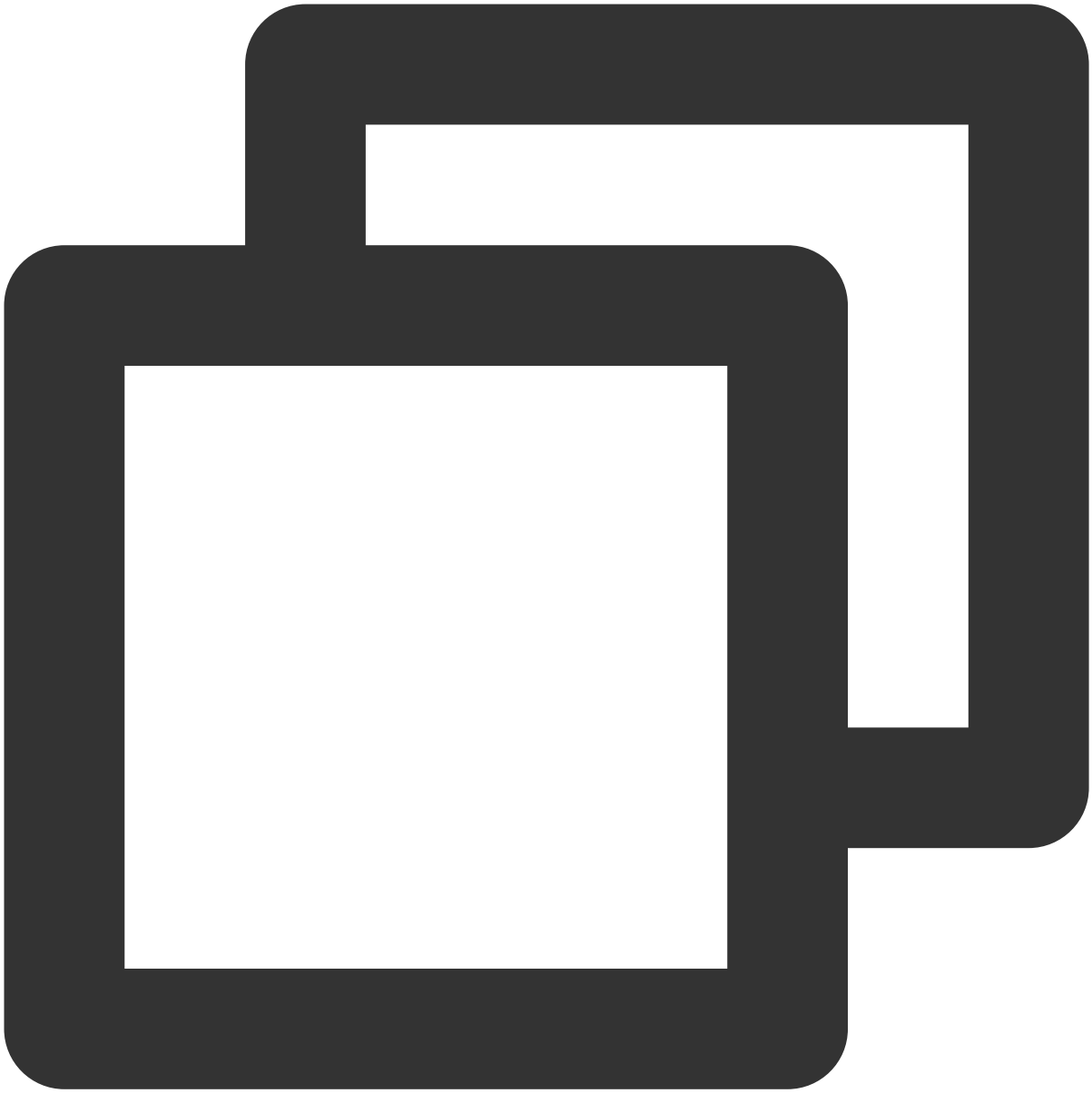
```
TencentEffectApi.getApi()
     ?.setOnCreateXmagicApiErrorListener((errorMsg, code) {
       TXLog.printlog("美顔オブジェクトの作成にエラーが発生しました errorMsg = $errorMsg ,
 });    ///美顔を作成する前に設定が必要です
```

顔、ジェスチャー、体の検出状態コールバックの設定



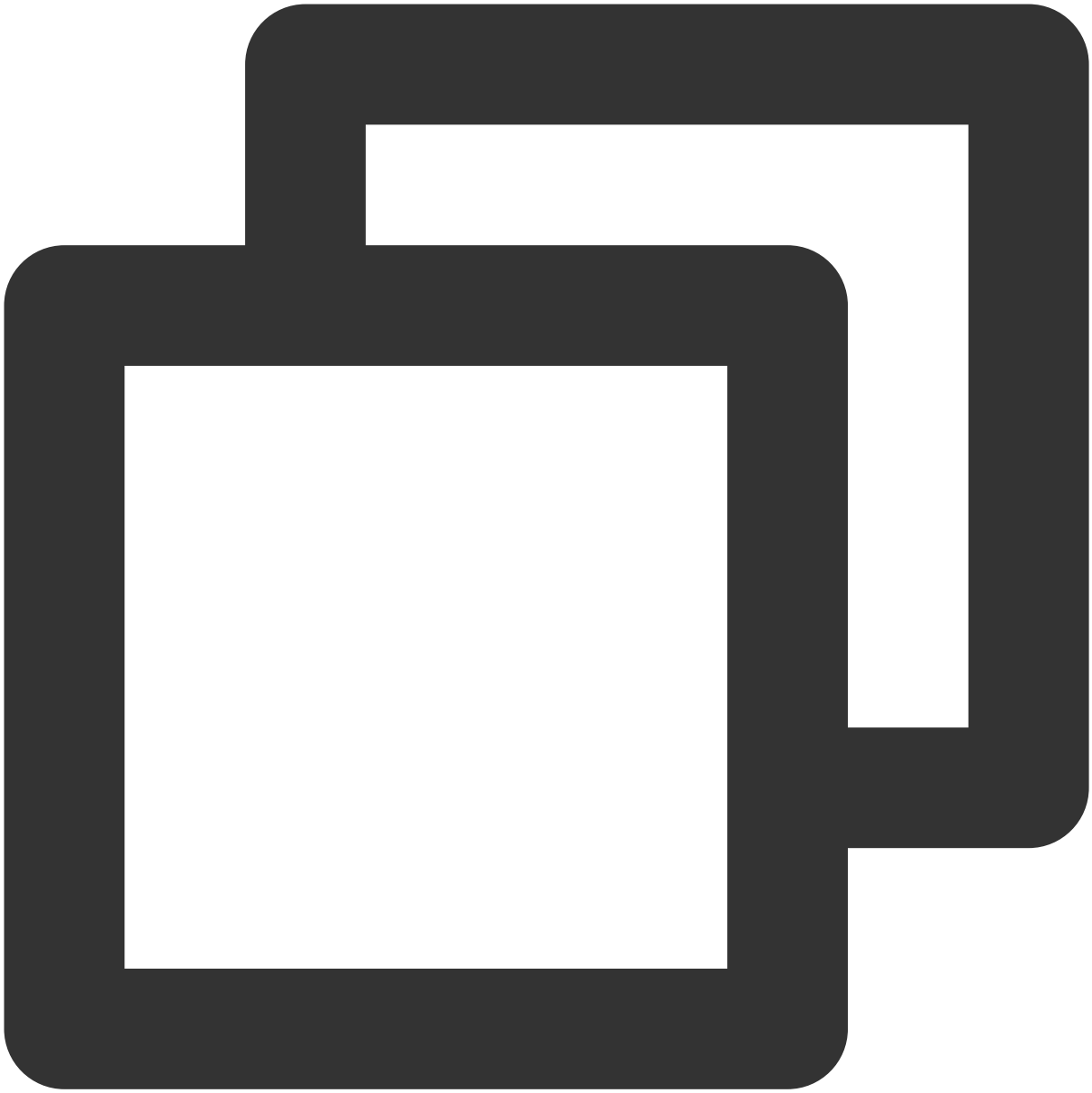
```
TencentEffectApi.getApi()?.setAIDataListener(XmagicAIDataListenerImp());
```

動的エフェクトプロンプトのコールバック関数の設定



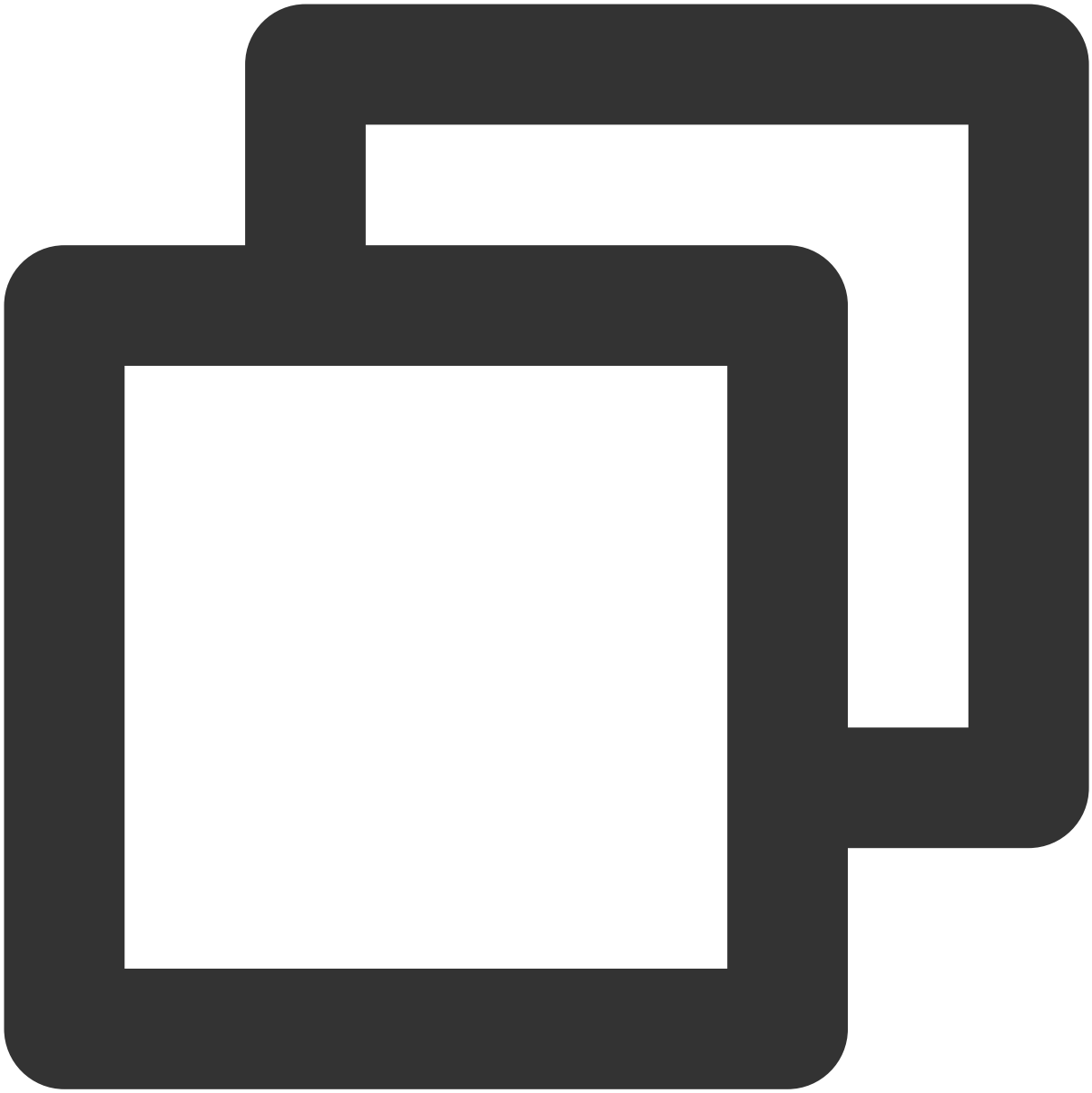
```
TencentEffectApi.getApi()?.setTipsListener(XmagicTipsListenerImp());
```

顔の特徴点位置情報などのデータコールバックを設定します（S1-05およびS1-06パッケージのみコールバックあり）



```
TencentEffectApi.getApi()?.setYTDataListener((data) {
   TXLog.printlog("setYTDataListener  $data");
 });
```

すべてのコールバックの削除。ページが破棄された場合はすべてのコールバックを削除する必要があります。



```
TencentEffectApi.getApi()?.setOnCreateXmagicApiErrorListener(null);
TencentEffectApi.getApi()?.setAIDataListener(null);
TencentEffectApi.getApi()?.setYTDataListener(null);
TencentEffectApi.getApi()?.setTipsListener(null);
```

説明

インターフェースの詳細についてはインターフェースドキュメントを、その他についてはDemoプロジェクトをそれぞれ参照できます。

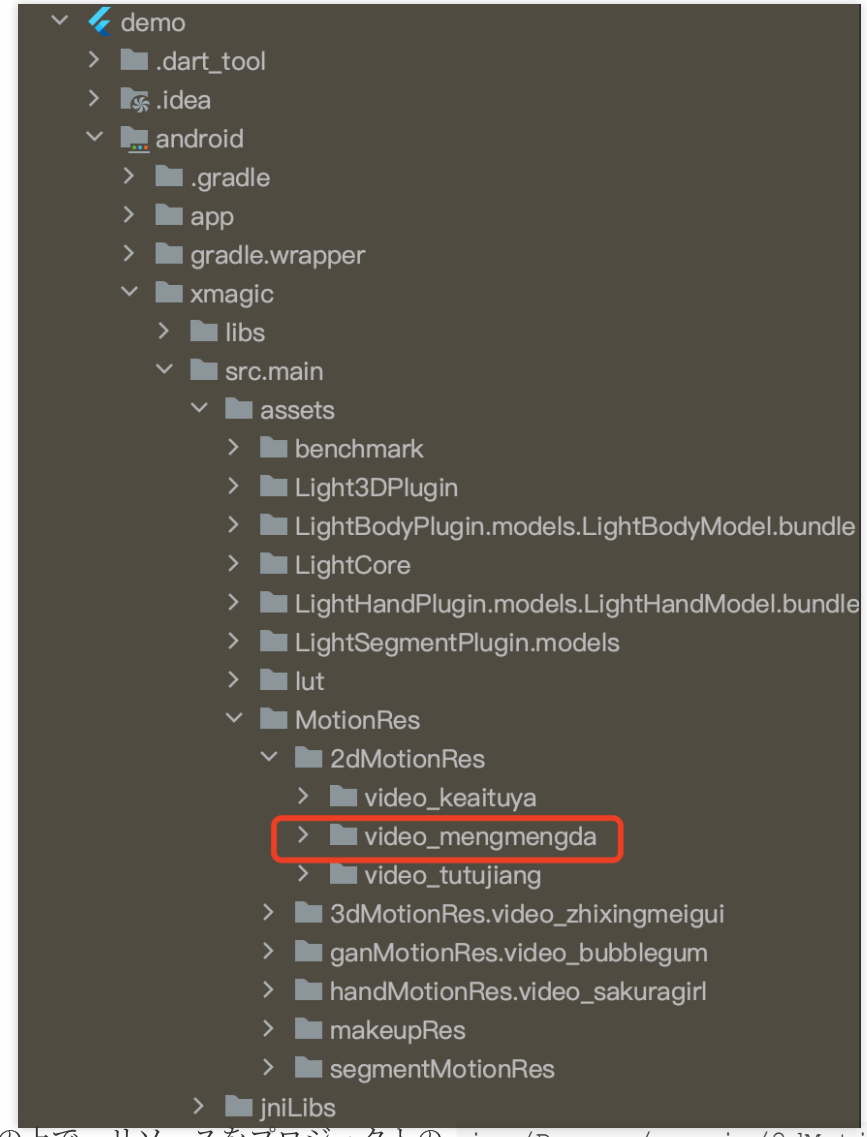
ステップ9：美顔パネル上の美顔データの追加と削除

BeautyDataManager、BeautyPropertyProducer、BeautyPropertyProducerAndroid、BeautyPropertyProducerIOSの4つのクラスでは、美顔パネルのデータ設定を自由に操作できます。

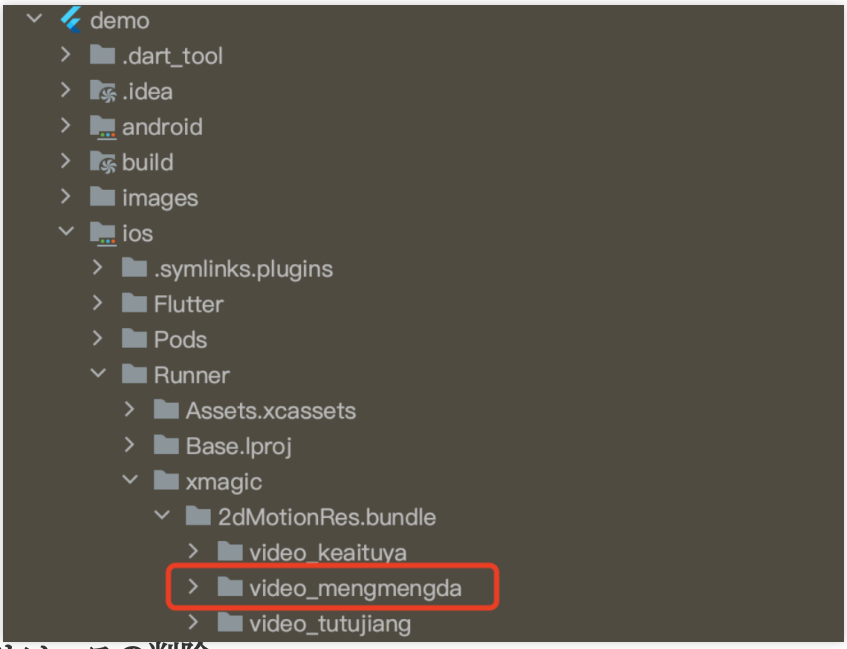
美顔リソースの追加

ステップ1の方法に従ってリソースファイルを対応するリソースフォルダ内に追加します。例えば、2D動的エフェクトのリソースを追加したい場合は次のようになります。

1. リソースを必ずプロジェクトの `android/xmagic/src.mian/assets/MotionRes/2dMotionRes` ディレクトリ下に置きます。



2. その上で、リソースをプロジェクトの `ios/Runner/xmagic/2dMotionRes.bundle` ディレクトリ下に追加します。



美顔リソースの削除

Licenseによっては美顔と美ボディの一部機能の権限がない場合があります。この一部機能は美顔パネル上に表示する必要はないため、美顔パネルデータの設定でこの一部機能の設定を削除する必要があります。
例えばリップのエフェクトを削除する場合は、BeautyPropertyProducerAndroidクラスおよびBeautyPropertyProducerIOSクラスのgetBeautyDataメソッドから次のコードをそれぞれ削除します。

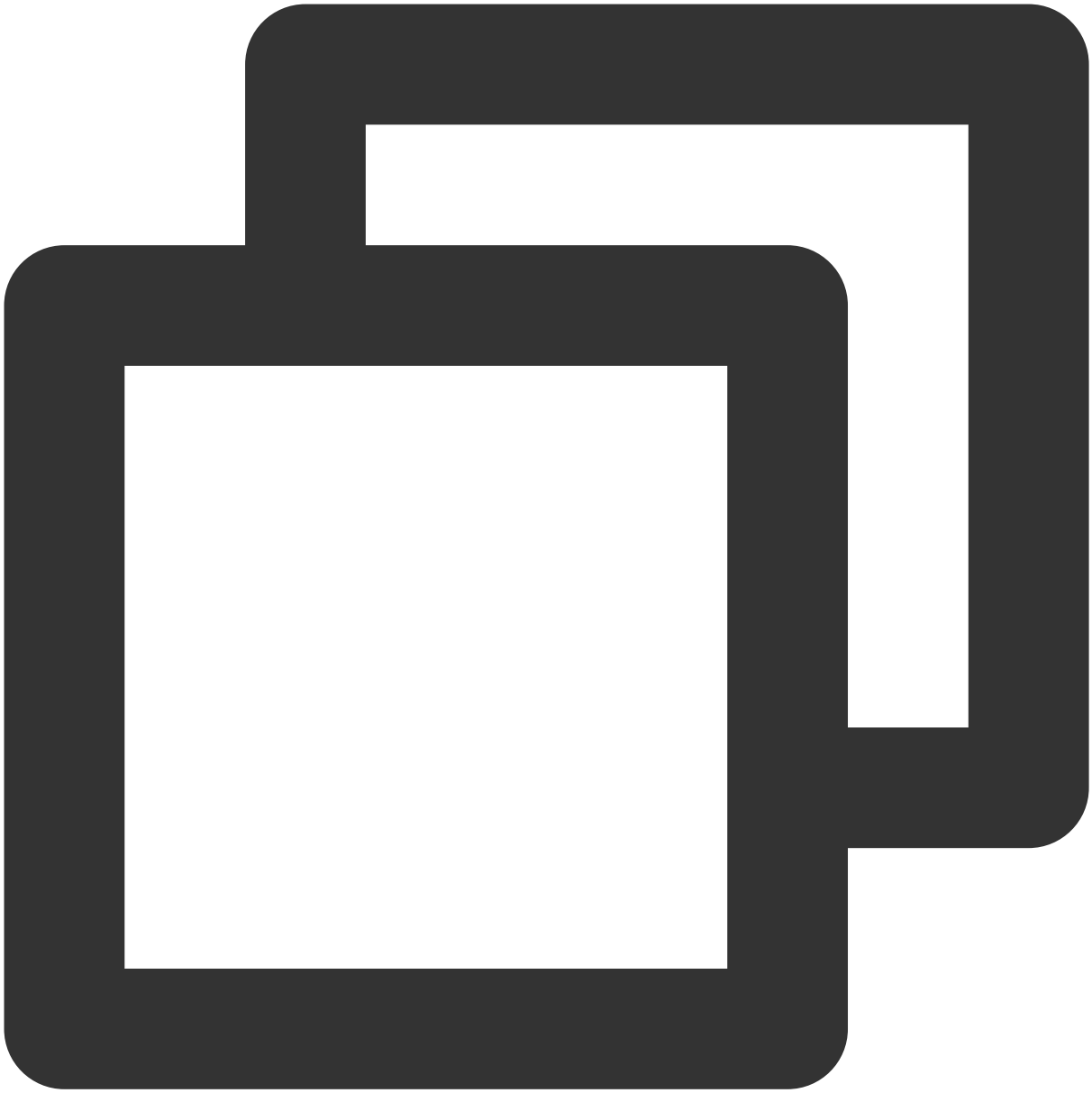
```
// Map<String, String> lipsResPathNames = {};
// lipsResPathNames["lips_fuguhong.png"] = "复古红";
// lipsResPathNames["lips_mitaose.png"] = "蜜桃色";
// lipsResPathNames["lips_shanhuju.png"] = "珊瑚橘";
// lipsResPathNames["lips_wenroufen.png"] = "温柔粉";
// lipsResPathNames["lips_huolicheng.png"] = "活力橙";
// List<XmagicUIProperty> itemLipsPropertys = [];
// String lipId = "beauty.lips.lipsMask";
// for (String ids in lipsResPathNames.keys) {
//   itemLipsPropertys.add(XmagicUIProperty(
//       uiCategory: Category.BEAUTY,
//       displayName: lipsResPathNames[ids]!,
//       id: lipId,
//       resPath: resPaths + ids,
//       thumbDrawableName: "beauty_lips",
//       effKey: BeautyConstant.BEAUTY_MOUTH_LIPSTICK,
//       effValue: XmagicPropertyValues(0, 100, 50, 0, 1),
//       rootDisplayName: "口红"));
// }
// XmagicUIProperty itemLips = XmagicUIProperty(
//     displayName: "口红",
//     thumbDrawableName: "beauty_lips",
//     uiCategory: Category.BEAUTY);
// itemLips.xmagicUIPropertyList = itemLipsPropertys;
// beautyList.add(itemLips);
```

Tencent EffectをUGSV SDKに統合
iOS

最終更新日：：2023-02-27 14:27:24

統合の準備

1. Demoパッケージをダウンロードして解凍し、Demoプロジェクトの `demo/XiaoShiPin/` ディレクトリ下のxmagickitフォルダをご自身のプロジェクトのpodfileファイルと同じレベルのディレクトリ下にコピーします。
2. Podfileファイル内に以下の依存関係を追加します。その後、 `pod install` コマンドを実行すると、インポートが完了します。



```
pod 'xmagickit', :path => 'xmagickit/xmagickit.podspec'
```

3. Bundle IDを、テスト用に申請した権限と同じものに変更します。

開発者環境要件

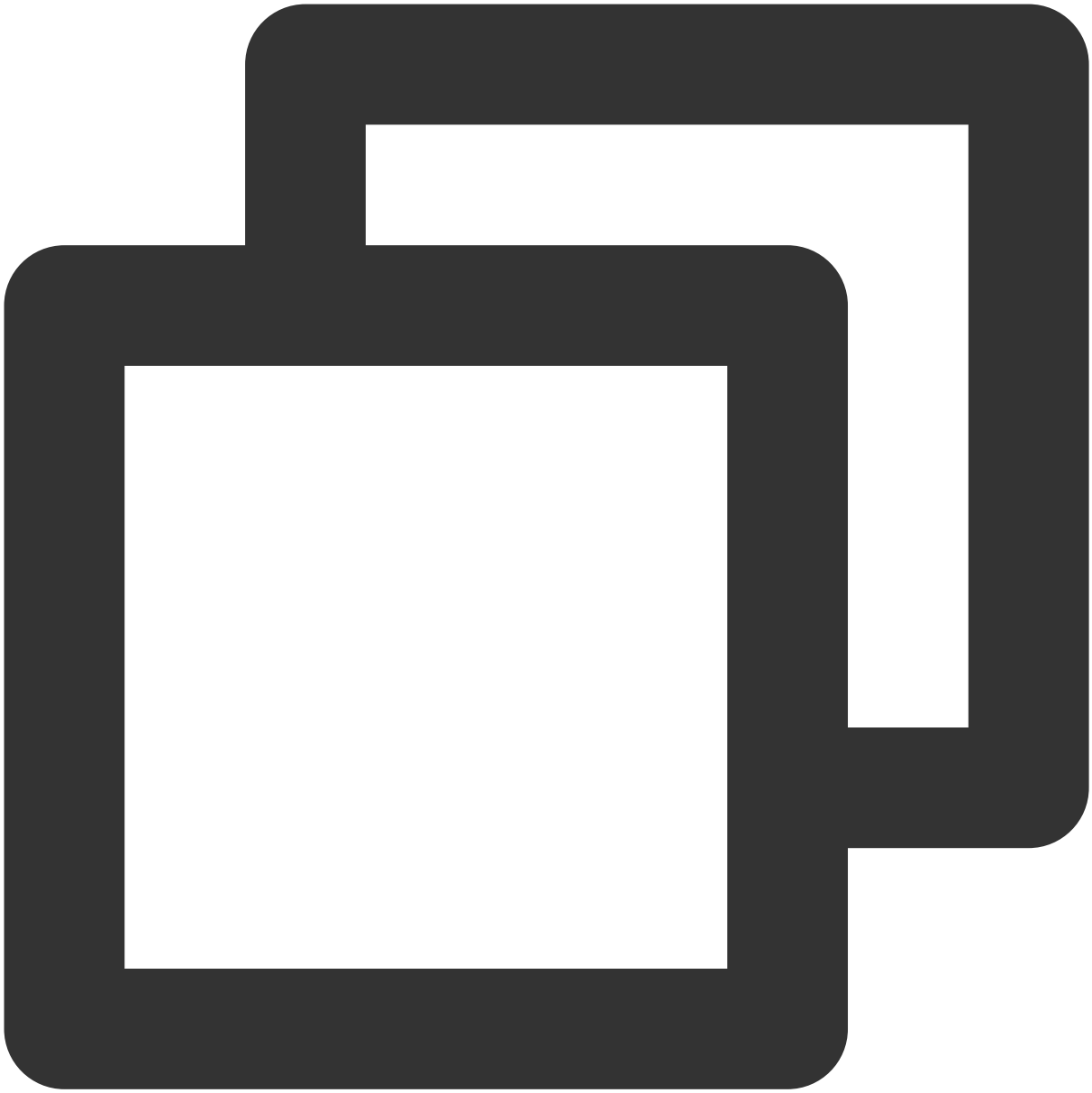
開発ツール XCode 11およびそれを超える必要があります。App Storeまたはダウンロードアドレスをクリックします。
推奨実行環境：

デバイス要件：iPhone 5およびそれ以上である必要があります。iPhone 6およびそれ未満の場合は、フロントカメラのサポートを最大720pとし、1080pはサポートしていません。
システム要件：iOS 12.0およびそれ以降のバージョン。

SDKインターフェースの統合

ステップ1：権限の初期化

プロジェクト `AppDelegate` の `didFinishLaunchingWithOptions` に次のコードを追加します。 `LicenseURL` 、 `LicenseKey` はTencent Cloud公式サイトに権限承認を申請した際の情報とします（XMagic SDKのバージョンが2.5.1より古い場合、 `TELicenseCheck.h` は、 `XMagic.framework` にあります。XMagicSDK のバージョンが2.5.1およびそれ以降の場合、 `TELicenseCheck.h` は、 `YTCommonXMagic.framework` にあります）。

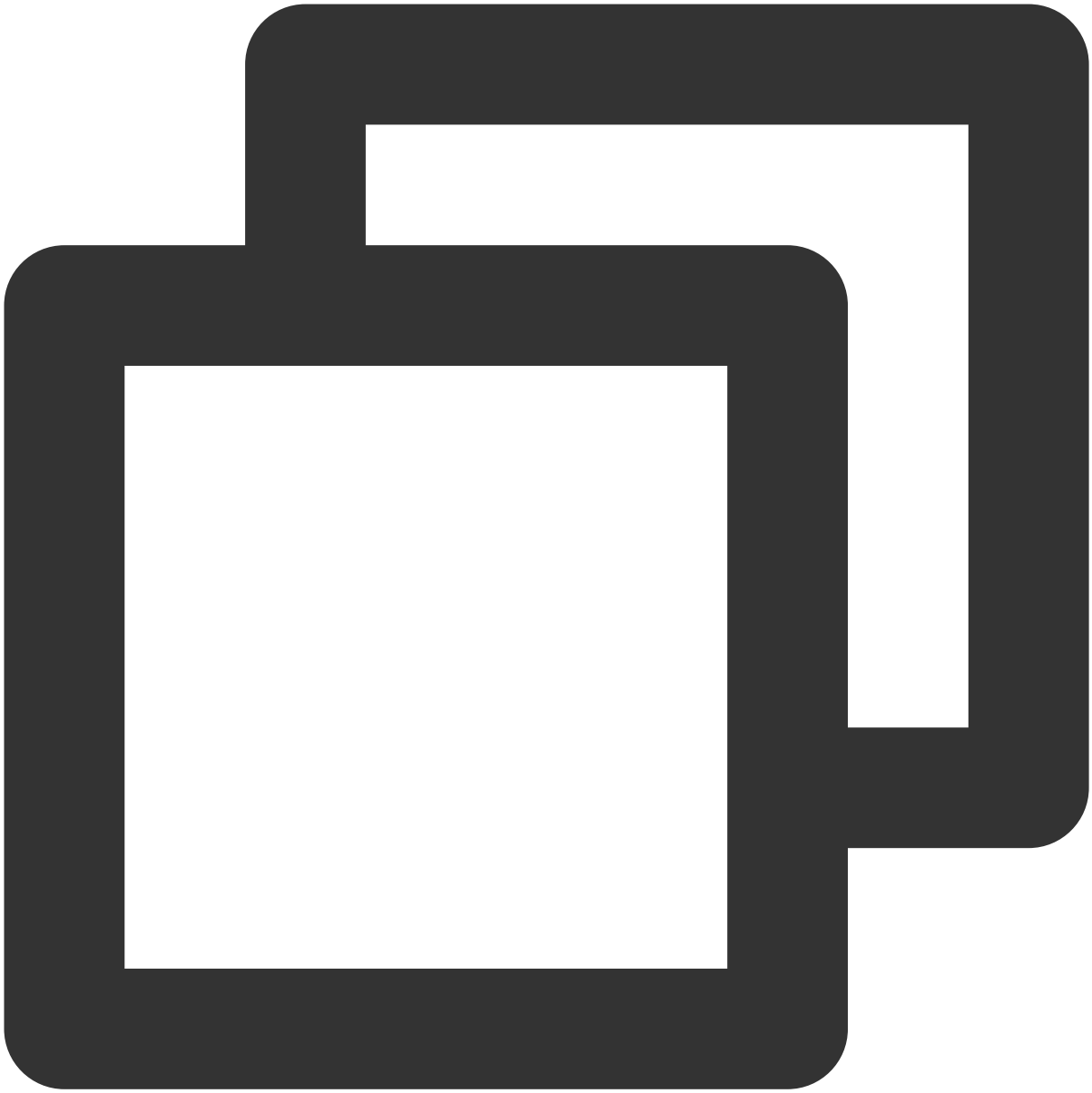


```
[TXUGCBase setLicenceURL:LicenseURL key:LicenseKey];

[TELicenseCheck setTELicense:LicenseURLkey:LicenseKey completion:^(NSInteger authre
              if (authresult == TELicenseCheckOk) {
                   NSLog(@"認証成功");
               } else {
                   NSLog(@"認証失敗");
               }
       }];
```

認証errorCode説明：

エラーコード	説明
0	成功です。Success
-1	入力パラメータが無効です（例：URLまたはKEYが空など）
-3	ダウンロードの段階で失敗しました。ネットワークの設定を確認してください
-4	ローカルから読み取ったTE権限承認情報が空です。IOの失敗による可能性があります。
-5	読み取ったVCUBE TEMP Licenseファイルの内容が空です。IOの失敗による可能性があります
-6	v_cube.licenseファイルのJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-7	署名の検証に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-8	復号に失敗しました。Tencent Cloudチームに連絡して処理を依頼してください
-9	TELicenseフィールド内のJSONフィールドが正しくありません。Tencent Cloudチームに連絡して処理を依頼してください
-10	ネットワークから解析したTE権限承認情報が空です。Tencent Cloudチームに連絡して処理を依頼してください
-11	TE権限承認情報をローカルファイルに書き込む際に失敗しました。IOの失敗による可能性があります
-12	ダウンロードに失敗しました。ローカルassetの解析も失敗しました
-13	認証に失敗しました
その他	Tencent Cloudチームに連絡して処理を依頼してください

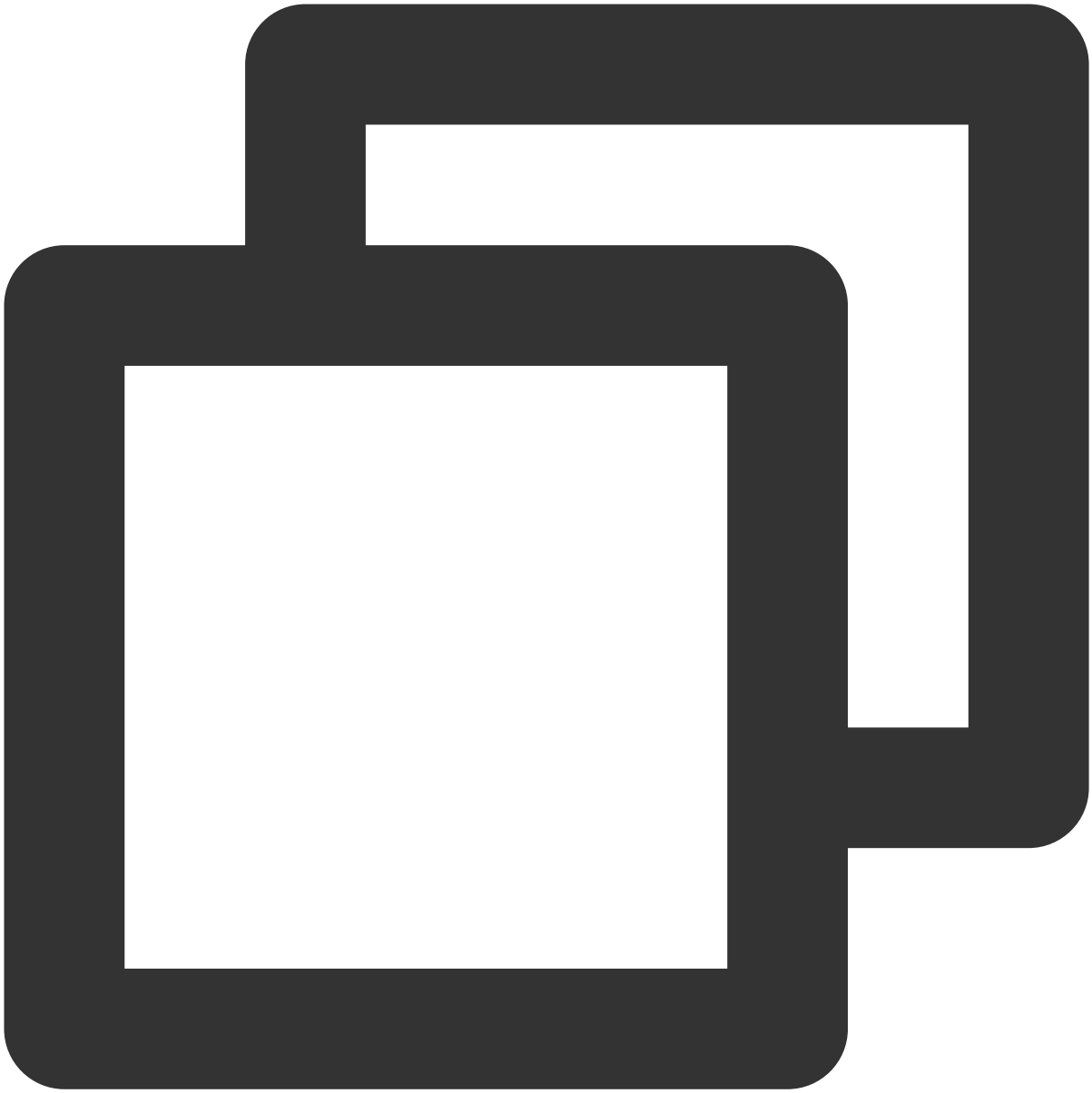
ステップ2：SDK素材リソースパスの設定



```
CGSize previewSize = [self getPreviewSizeByResolution:self.currentPreviewResolution
NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirecto
beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_config
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] && !isD
    NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beautyConfig
    NSError *jsonError;
    NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8StringEncodin
    beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData
```

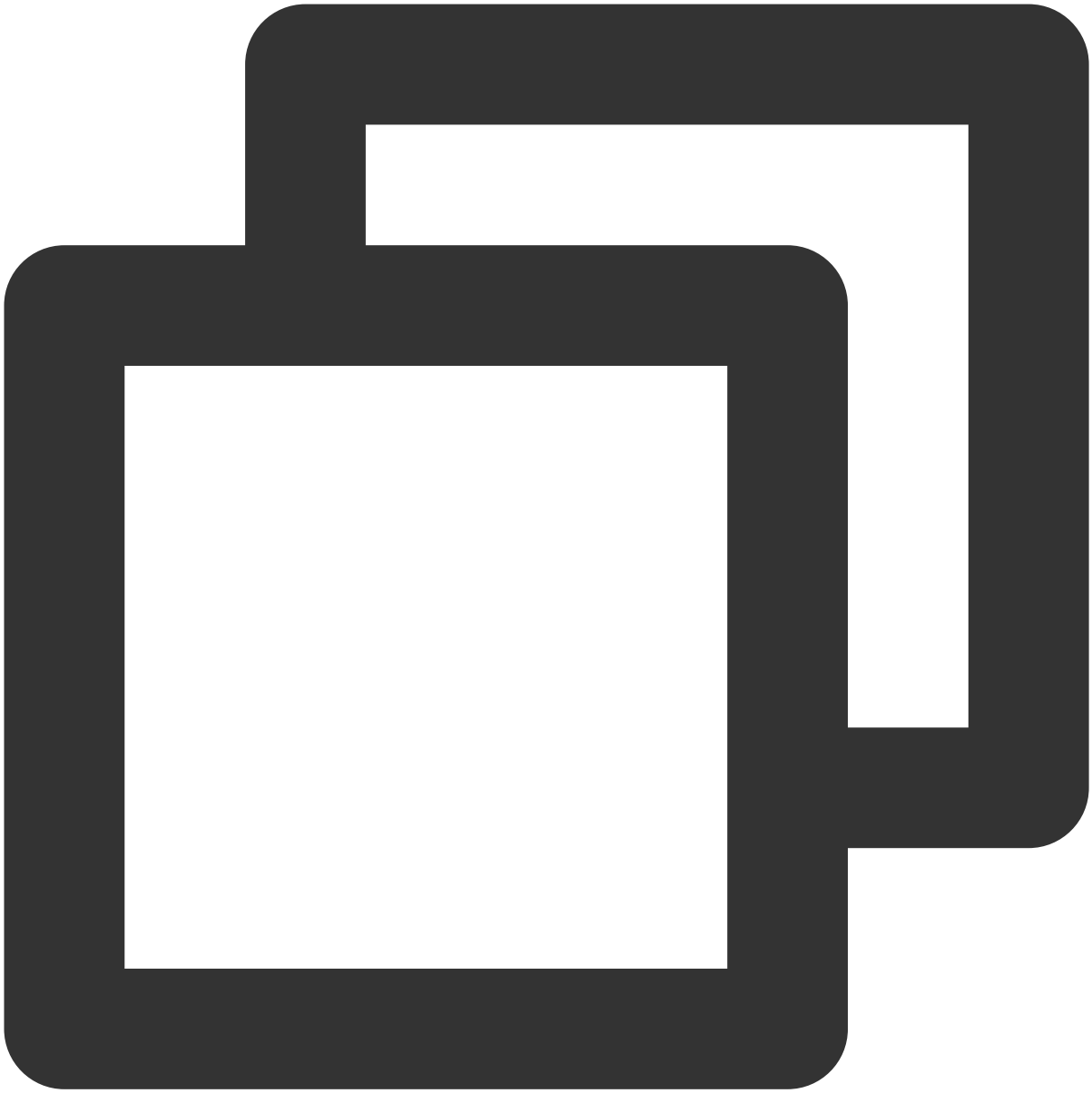
```
                                                options:NSJSONReadingMutableContainers
                                                error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle] bundlePath],
                             @"tnn_"
                             @"beauty_config":beautyConfigJson
};
// Init beauty kit
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDi
```

ステップ3：ログおよびイベント監視の追加



```
// Register log
[self.beautyKit registerSDKEventListener:self];
[self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL];
```

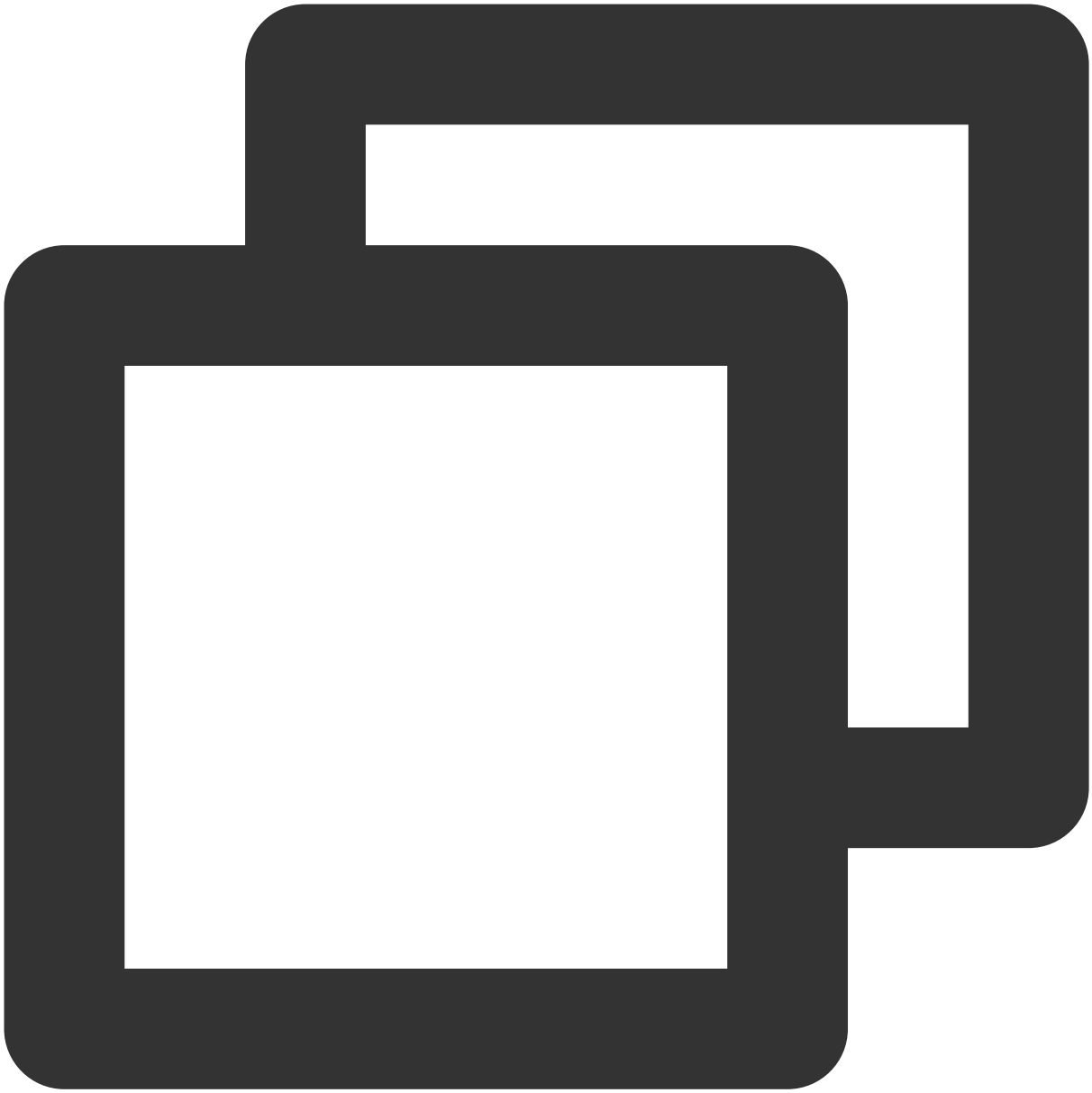
ステップ4：美顔の各種効果の設定



```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:(NSString *
```

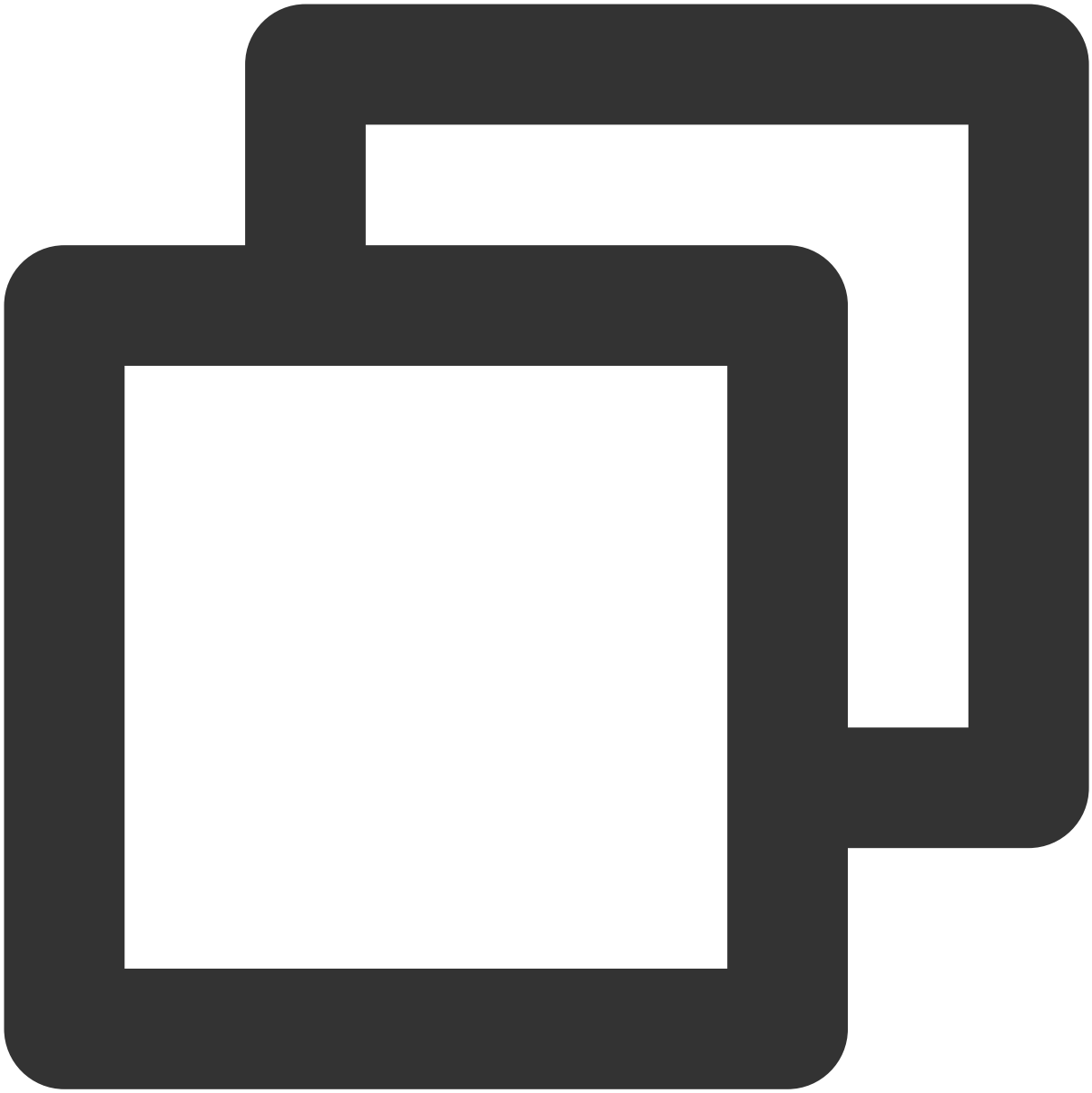
ステップ5：レンダリング処理の実施

UGSVの前処理フレームコールバックインターフェースで、YTProcessInputを作成してtextureIdをSDKに渡し、レンダリング処理を行います。



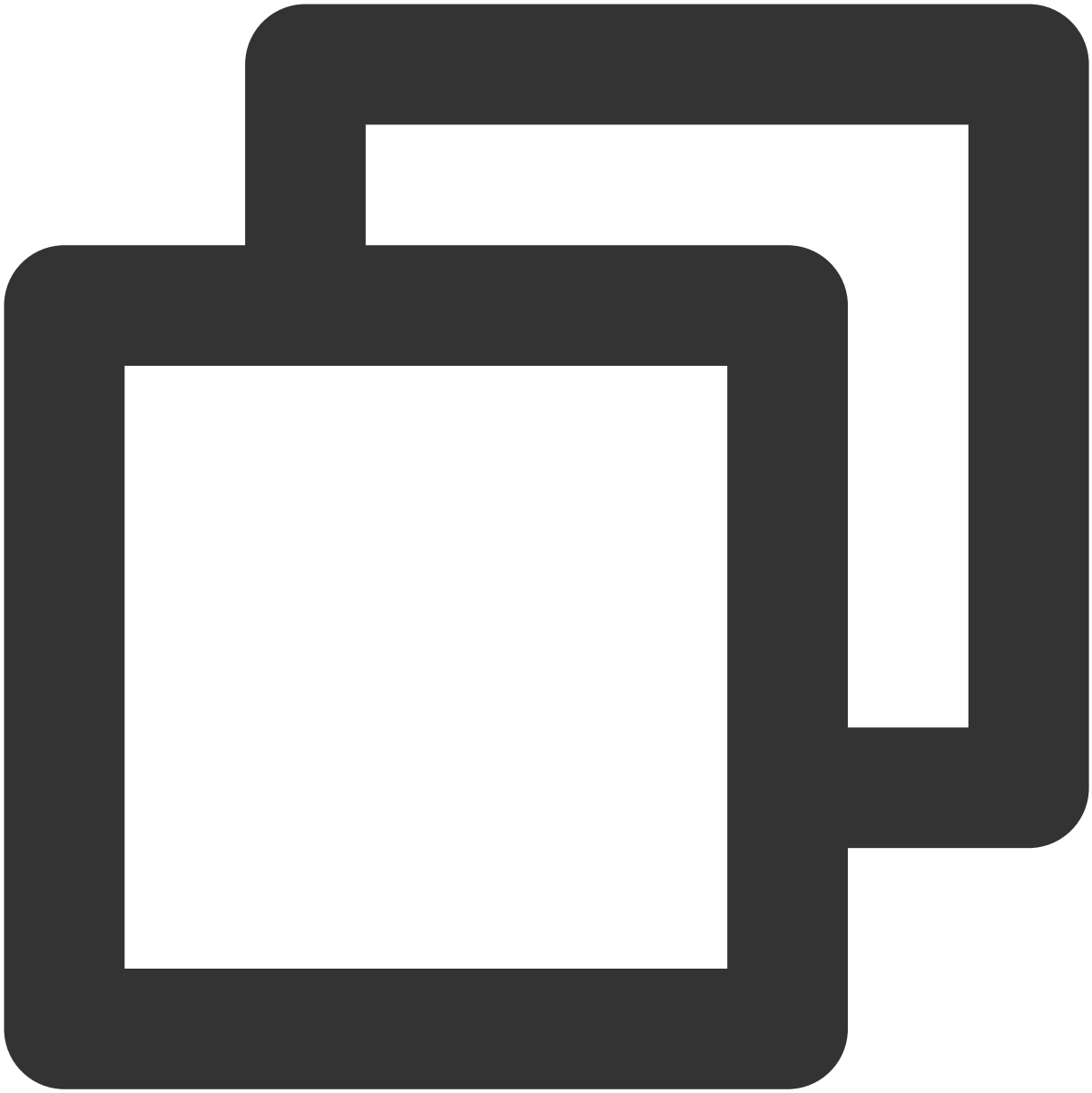
```
[self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft withOrientat
```

ステップ6：SDKの一時停止/再開



```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

ステップ7：レイアウトにSDK美顔パネルを追加



```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
if (@available(iOS 11.0, *)) {
    gSafeInset = [UIApplication sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
    {
        gSafeInset = UIEdgeInsetsZero;
    }
```

```
dispatch_async(dispatch_get_main_queue(), ^{
    //美顔オプションインターフェース
    _vBeauty = [[BeautyView alloc] init];
    [self.view addSubview:_vBeauty];
    [_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
        make.width.mas_equalTo(self.view);
        make.centerX.mas_equalTo(self.view);
        make.height.mas_equalTo(254);
        if(gSafeInset.bottom > 0.0){  // 全画面適用
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
        } else {
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
        }
    }];
        _vBeauty.hidden = YES;
});
```

Android

最終更新日：：2022-12-15 11:30:53

手順1：Demoプロジェクトの解凍

1. Tencent Effect TEを統合したUGSV Demoプロジェクトをダウンロードします。このDemoは、Tencent Effect SDK S1-04パッケージに基づいて作成されています。
2. リソースの置き換え：このDemoプロジェクトで使用するSDKパッケージが実際のパッケージと一致するとは限らないため、このDemoの関連SDKファイルを実際に使用されるパッケージのSDKファイルに置き換える必要があります。具体的な操作は次のとおりです。
   - `xmagickit module` の `build.gradle` ファイルで下記を見つけます。

```
api 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
```

購入した依存パッケージに置き換えます。

- パッケージに動的エフェクトおよびフィルター機能が含まれる場合は、Tencent Effect SDKダウンロードページで対応するリソースをダウンロードし、動的エフェクトおよびフィルター素材を `xmagickit module` 下の以下のディレクトリに配置する必要があります。
  - 動的エフェクト： `../assets/MotionRes`
  - フィルター： `../assets/lut`

3. Demoプロジェクトのxmagickitモジュールを実際のプロジェクトにインポートします。

手順2：appモジュールのbuild.gradleを開く

applicationIdを、申請したテスト権限と同じパッケージ名に変更します。

手順3：SDKインターフェースの統合

DemoプロジェクトのUGCKitVideoRecordタイプを参照できます。

1. **権限承認：**

```
//認証の注意事項およびエラーコードの詳細については、https://www.tencentcloud.com/document/product/1143/45385#.E6.AD.A5.E9.AA.A4.E4.B8.80.EF.BC.9A.E9.89.B4.E6.9D.83をご参照ください
XMagicImpl.checkAuth(new TELicenseCheck.TELicenseCheckListener() {
@Override
public void onLicenseCheckFinish(int errorCode, String msg) {
if (errorCode == TELicenseCheck.ERROR_OK) {
loadXmagicRes();
}else{
Log.e("TAG", "auth fail ,please check auth url and key" + errorCode + " " + msg);
}
}
});
```

2. 素材の初期化：

```
private void loadXmagicRes() {
if (XMagicImpl.isLoadedRes) {
XmagicResParser.parseRes(mActivity.getApplicationContext());
initXMagic();
return;
}
new Thread(new Runnable() {
@Override
public void run() {
XmagicResParser.copyRes(mActivity.getApplicationContext());
XmagicResParser.parseRes(mActivity.getApplicationContext());
XMagicImpl.isLoadedRes = true;
new Handler(Looper.getMainLooper()).post(new Runnable() {
@Override
public void run() {
initXMagic();
}
});
}
}).start();
}
```

3. UGSVと美顔のバインド：

```
private void initBeauty() {
TXUGCRecord instance = TXUGCRecord.getInstance(UGCKit.getAppContext());
```

```
instance.setVideoProcessListener(new TXUGCRecord.VideoCustomProcessListener() {
@Override
public int onTextureCustomProcess(int textureId, int width, int height) {
if (xmagicState == XMagicImpl.XmagicState.STARTED && mXMagic != null) {
return mXMagic.process(textureId, width, height);
}
return textureId;
}
@Override
public void onDetectFacePoints(float[] floats) {
}
@Override
public void onTextureDestroyed() {
if (Looper.getMainLooper() != Looper.myLooper()) { //メインスレッドではない
boolean stopped = xmagicState == XMagicImpl.XmagicState.STOPPED;
if (stopped || xmagicState == XMagicImpl.XmagicState.DESTROYED) {
if (mXMagic != null) {
mXMagic.onDestroy();
}
}
if (xmagicState == XMagicImpl.XmagicState.DESTROYED) {
TXUGCRecord.getInstance(UGCKit.getAppContext()).setVideoProcessListener(null);
}
}
}
});
}
```

4. **SDKの一時停止/破棄：**

`onPause()` は美顔効果の一時停止に使用し、Activity/Fragmentライフサイクルメソッドにおいて実行できます。onDestroyメソッドはGLスレッドで呼び出す必要があります（onTextureDestroyedメソッドでXMagicImplオブジェクトの `onDestroy()` を呼び出すことができます）。その他の使用については事例にあるonTextureDestroyedメソッドをご参照ください。

```
@Override
public void onTextureDestroyed() {
if (Looper.getMainLooper() != Looper.myLooper()) { //メインスレッドではない
boolean stopped = xmagicState == XMagicImpl.XmagicState.STOPPED;
if (stopped || xmagicState == XMagicImpl.XmagicState.DESTROYED) {
if (mXMagic != null) {
mXMagic.onDestroy();
}
}
if (xmagicState == XMagicImpl.XmagicState.DESTROYED) {
```

```
TXUGCRecord.getInstance(UGCKit.getAppContext()).setVideoProcessListener(null);
}
}
}
```

5. レイアウトに美顔パネルをロードしたレイアウトを追加：

```
<RelativeLayout
android:id="@+id/panel_layout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:visibility="gone"/>
```

6. 美顔オブジェクトを作成して美顔パネルを追加

```
private void initXMagic() {
if (mXMagic == null) {
mXMagic = new XMagicImpl(mActivity, getBeautyPanel());
}else{
mXMagic.onResume();
}
}
```

具体的な操作についてはDemoプロジェクトのUGCKitVideoRecordタイプをご参照ください。

Avatarバーチャルヒューマン統合ガイド
iOS
Avatarクイックアクセス

最終更新日：：2023-02-27 14:18:15

AvatarはTencent Effectの機能の一部であるため、先にTencent Effect美顔エフェクトSDKを統合してからAvatar素材をロードする必要があります。Tencent Effect美顔エフェクトSDKを接続していない場合は、Tencent Effectの独立した統合 >を参照して理解し、統合を行うことができます。

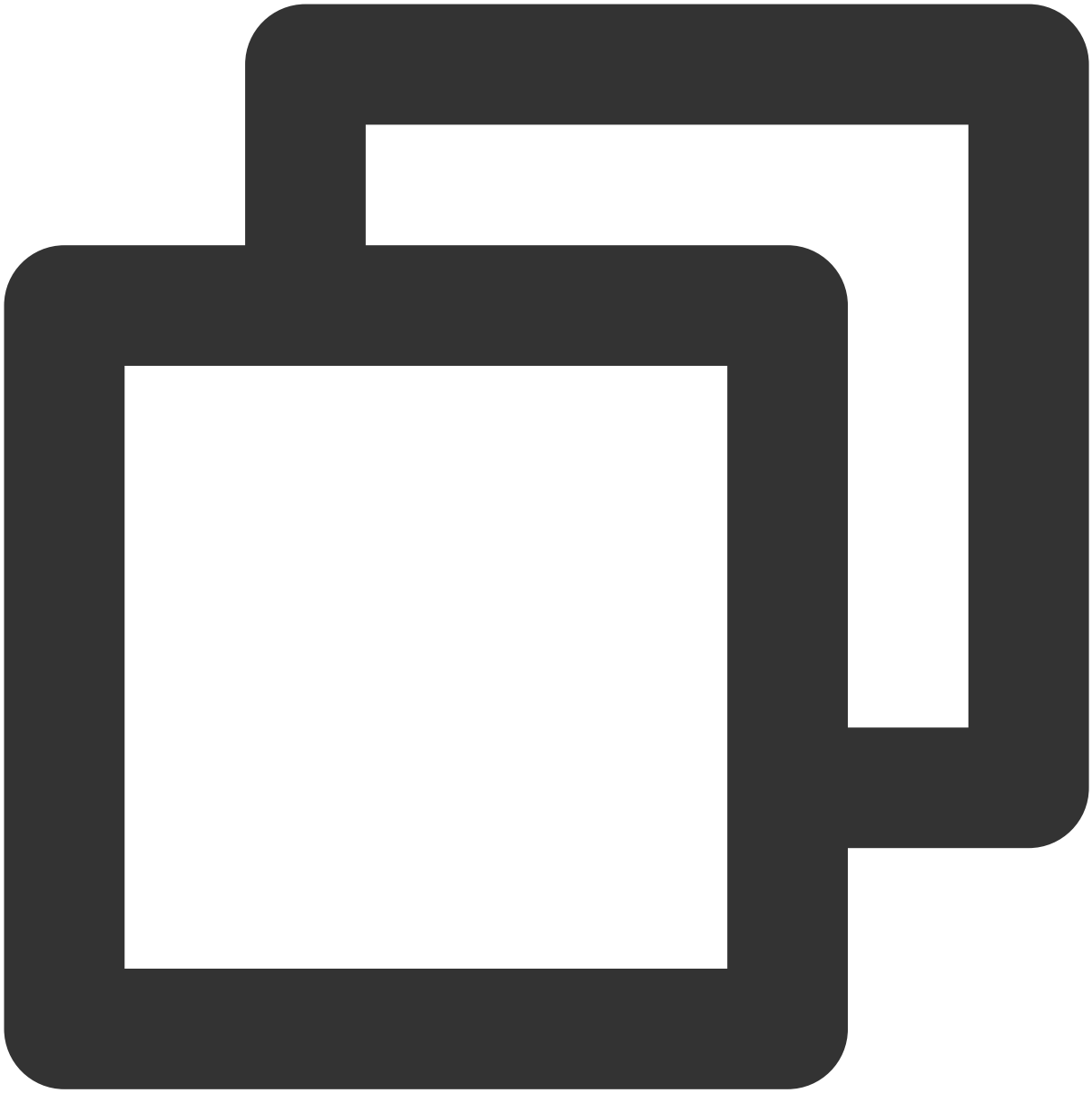
ステップ1：Avatar素材の準備

1. Tencent Effect SDKを統合します。
2. 対応するDemoプロジェクトを公式サイトからダウンロードし、解凍します。
3. Demo内の `BeautyDemo/bundle/avatarMotionRes.bundle` 素材ファイルをご自身のプロジェクトにコピーします。

ステップ2：Demoインターフェースの接続

接続方法

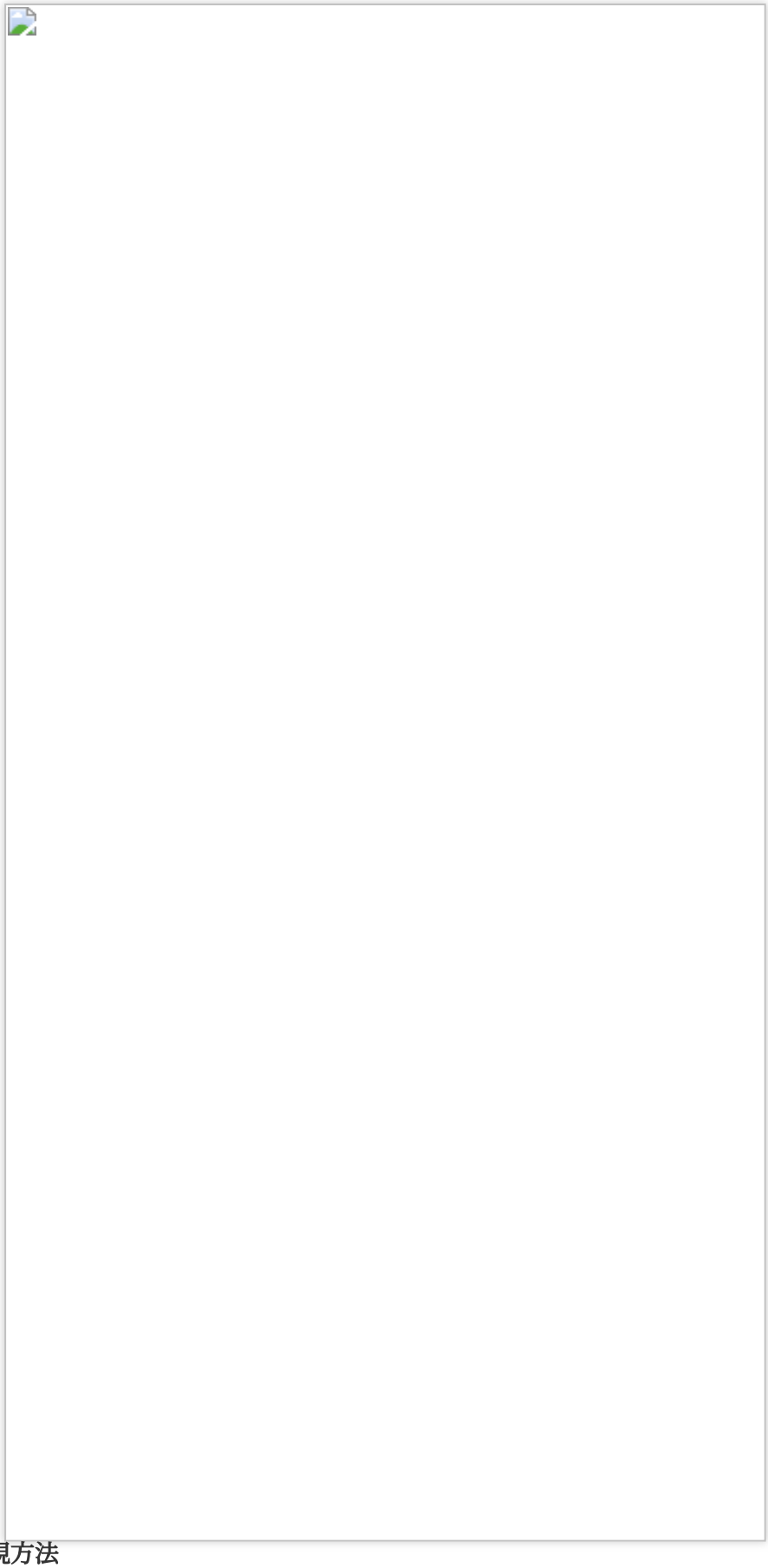
1. プロジェクトではBeautyDemoと同じAvatar操作インターフェースを使用します。
2. Demo内の `BeautyDemo/Avatar` フォルダ下のすべてのクラスをプロジェクトにコピーし、次のコードを追加すれば完了です。



```
AvatarViewController *avatarVC = [[AvatarViewController alloc] init];
avatarVC.modalPresentationStyle = UIModalPresentationFullScreen;
avatarVC.currentDebugProcessType = AvatarPixelData; // 画像またはテクスチャId方式
[self presentViewController:avatarVC animated:YES completion:nil];
```

Demoインターフェースの説明

1. Demo UI



2. 实现方法

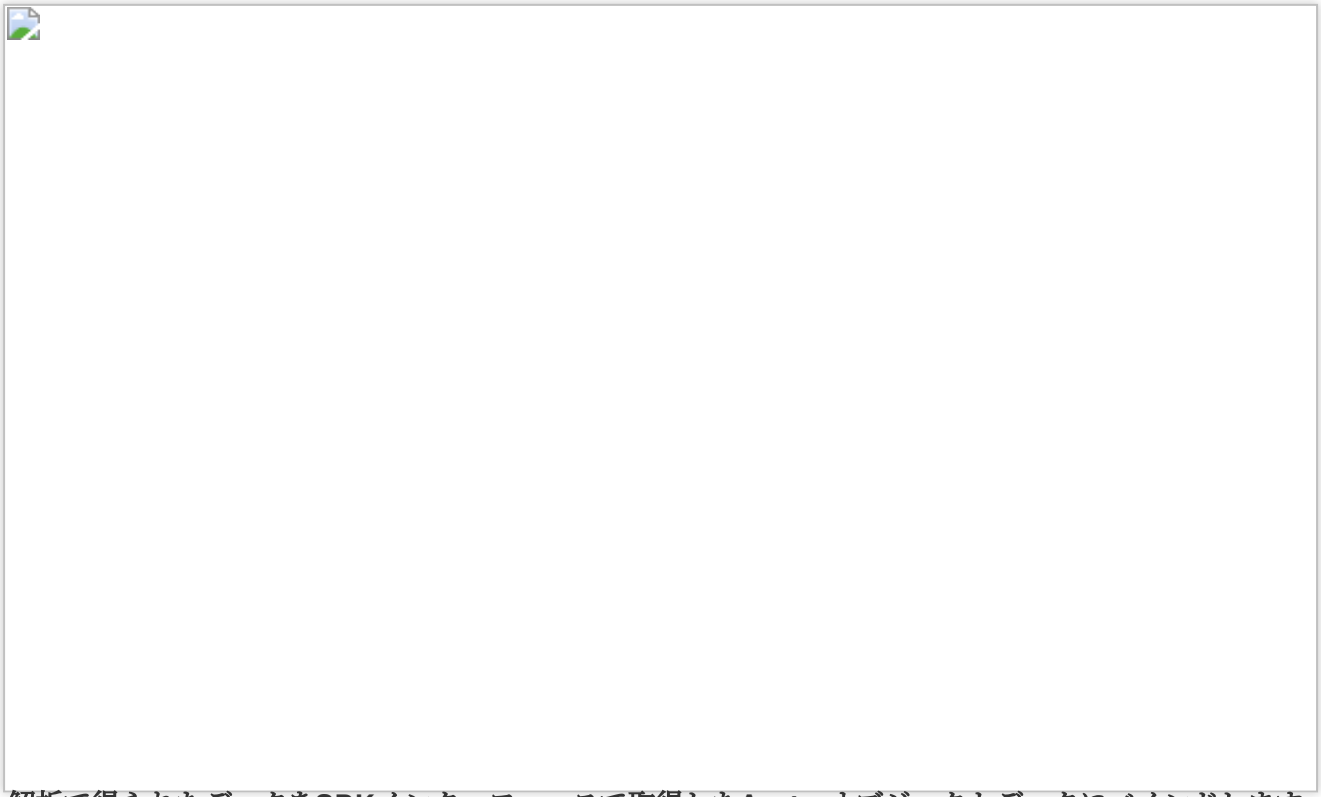
操作パネルデータはJSONファイルの解析によって取得したものです。Demo内のこのファイルを `BeautyDemo/Avatar/` ディレクトリに配置します。

JSON構造とUIパネルの対応関係

headは最初のiconで選択した内容です。

subTabsは右側の第2階層メニューに対応します。

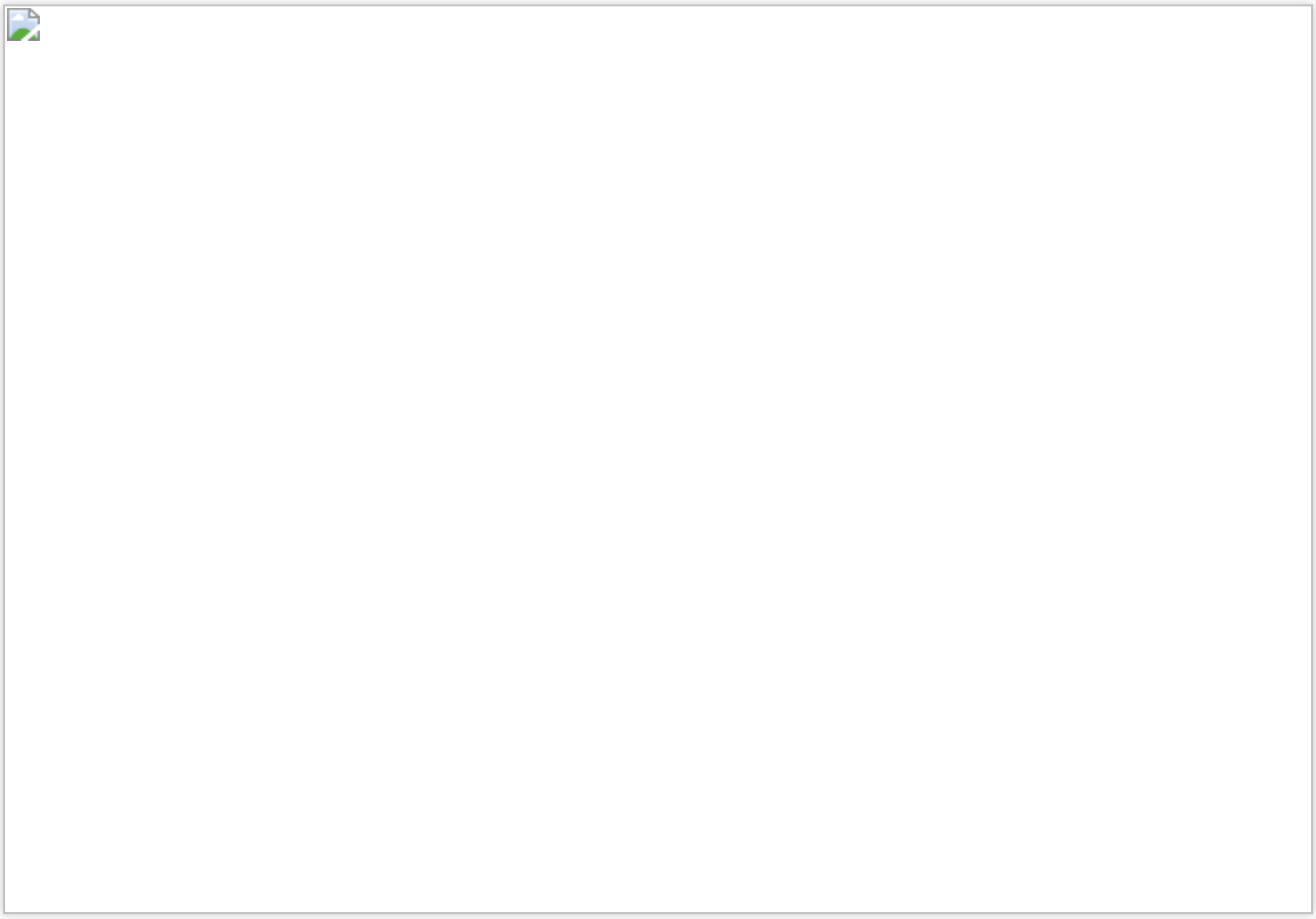
itemsは右側の第3階層メニューに対応します。



パネル解析で得られたデータをSDKインターフェースで取得したAvatarオブジェクトデータにバインドします

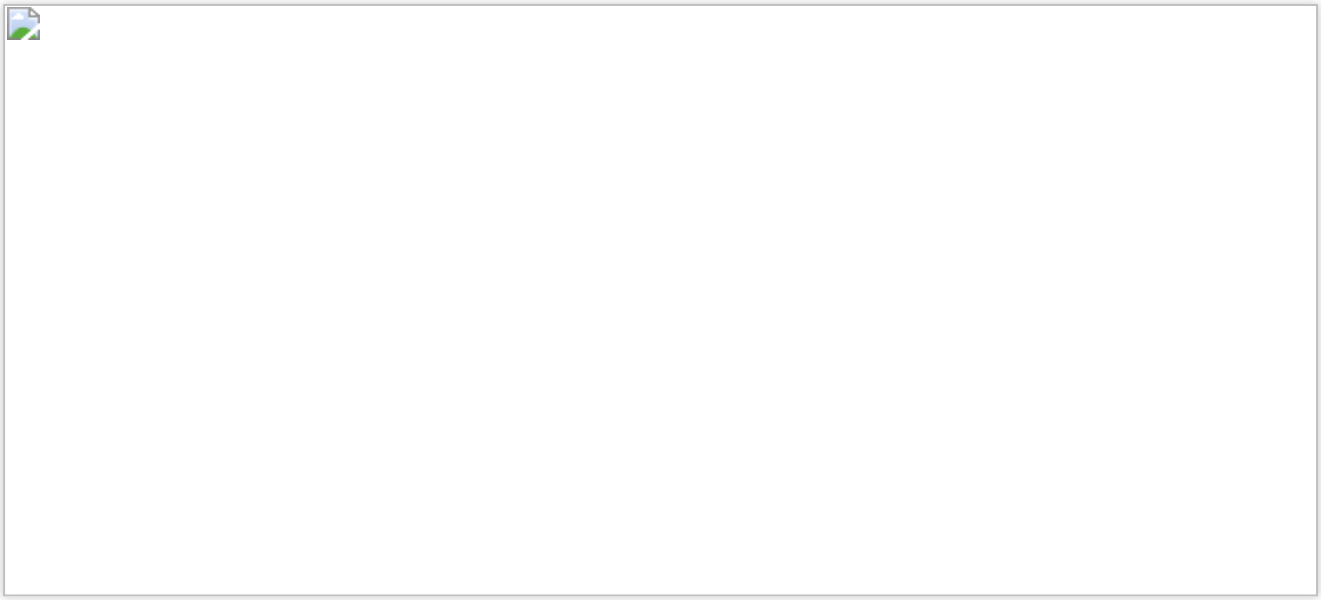
下図の上半分はSDKが取得したavatarディクショナリ（keyはcategory、valueはavatar配列）、下半分はパネルのデータです。パネルのオプションをクリックすると、パネルの第2階層タイトルからcategory（**赤枠で表示**）を取

得し、このcategoryによってSDKが返すavatarディクショナリ内で対応するavatarData配列を取得できます。パネルの第3階層タイトルからID（**青枠で表示**）を取得し、このIDによってavatarData配列の中で対応するavatarDataオブジェクトにマッチさせ、このオブジェクトをSDKのupdateAvatarに渡してアバター制作を完了することができます。



3. タイトルのアイコン/文字の変更

Demo UI画像に示すJSONファイルを変更します。例えば**ヘッダー**に表示するiconを変更するには、下図の赤枠内のiconUrlとcheckedIconUrlを変更します。



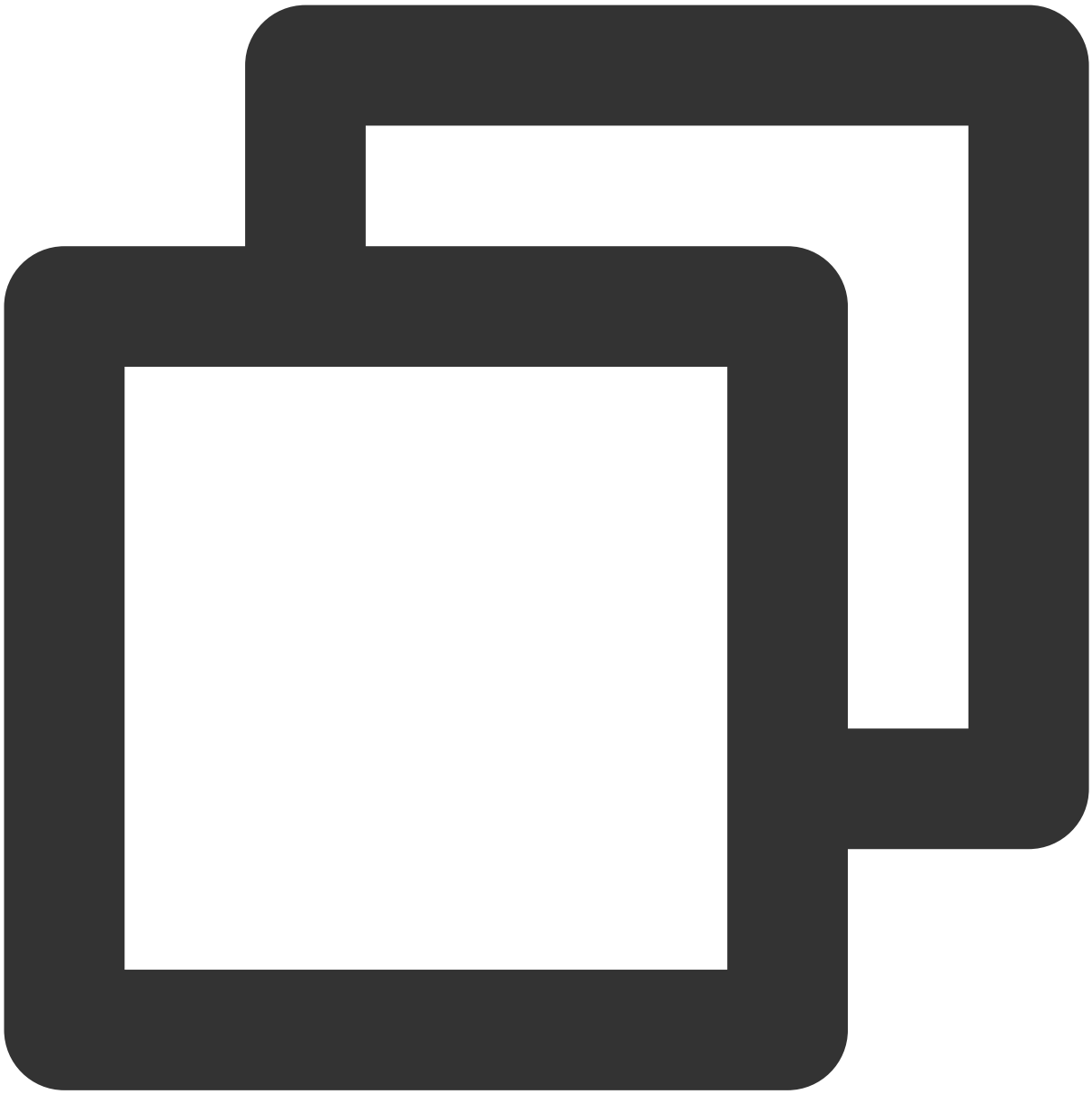
ステップ3：アバター制作機能のカスタマイズ

`BeautyDemo/Avatar/Controller` 内の**AvatarViewController**関連コードを参照できます。

説明

インターフェースの説明についてはAvatar SDKの説明をご参照ください。

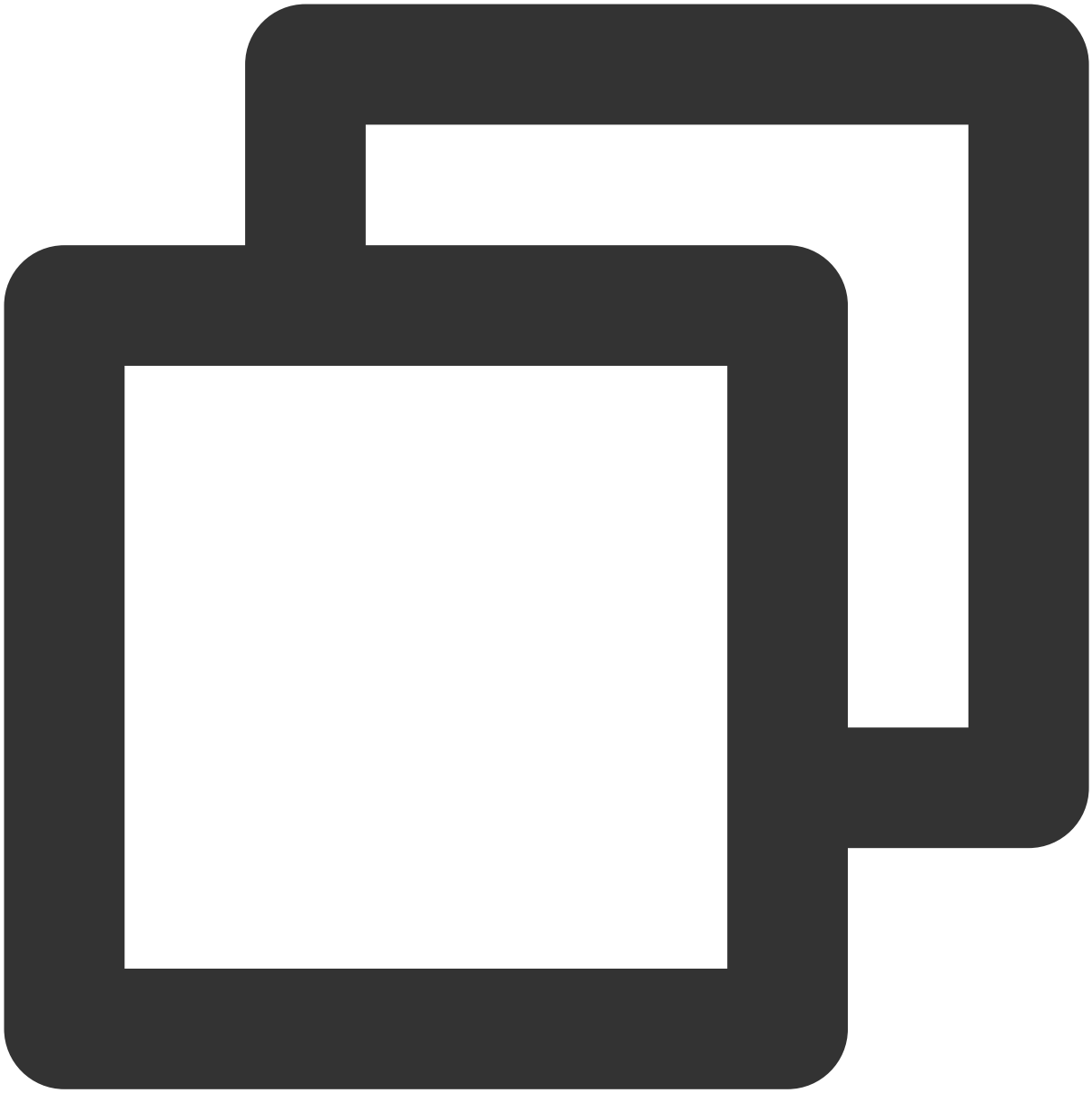
1. xmagicオブジェクトを作成し、Avatarデフォルトテンプレートを設定します。



```
- (void)buildBeautySDK {

    CGSize previewSize = CGSizeMake(kPreviewWidth, kPreviewHeight);
    NSString *beautyConfigPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDir
    beautyConfigPath = [beautyConfigPath stringByAppendingPathComponent:@"beauty_co
    NSFileManager *localFileManager=[[NSFileManager alloc] init];
    BOOL isDir = YES;
    NSDictionary * beautyConfigJson = @{};
    if ([localFileManager fileExistsAtPath:beautyConfigPath isDirectory:&isDir] &&
```

```
            NSString *beautyConfigJsonStr = [NSString stringWithContentsOfFile:beau
            NSError *jsonError;
            NSData *objectData = [beautyConfigJsonStr dataUsingEncoding:NSUTF8Strin
            beautyConfigJson = [NSJSONSerialization JSONObjectWithData:objectData

    }
    NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                                                                @"root_path":[[NSBundl
                                                                @"tnn_"
                                                                @"beauty_config":beaut
    };
    // Init beauty kit
    self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:asse
    // Register log
    [self.beautyKit registerSDKEventListener:self];
    [self.beautyKit registerLoggerListener:self withDefaultLevel:YT_SDK_ERROR_LEVEL

    // 素材ファイルに対応するパスを渡すとavatarデフォルトキャラクターをロードできます
    AvatarGender gender = self.genderBtn.isSelected ? AvatarGenderFemale : AvatarGe
    NSString *bundlePath = [self.resManager avatarResPath:gender];
    [self.beautyKit loadAvatar:bundlePath exportedAvatar:nil];

}
```

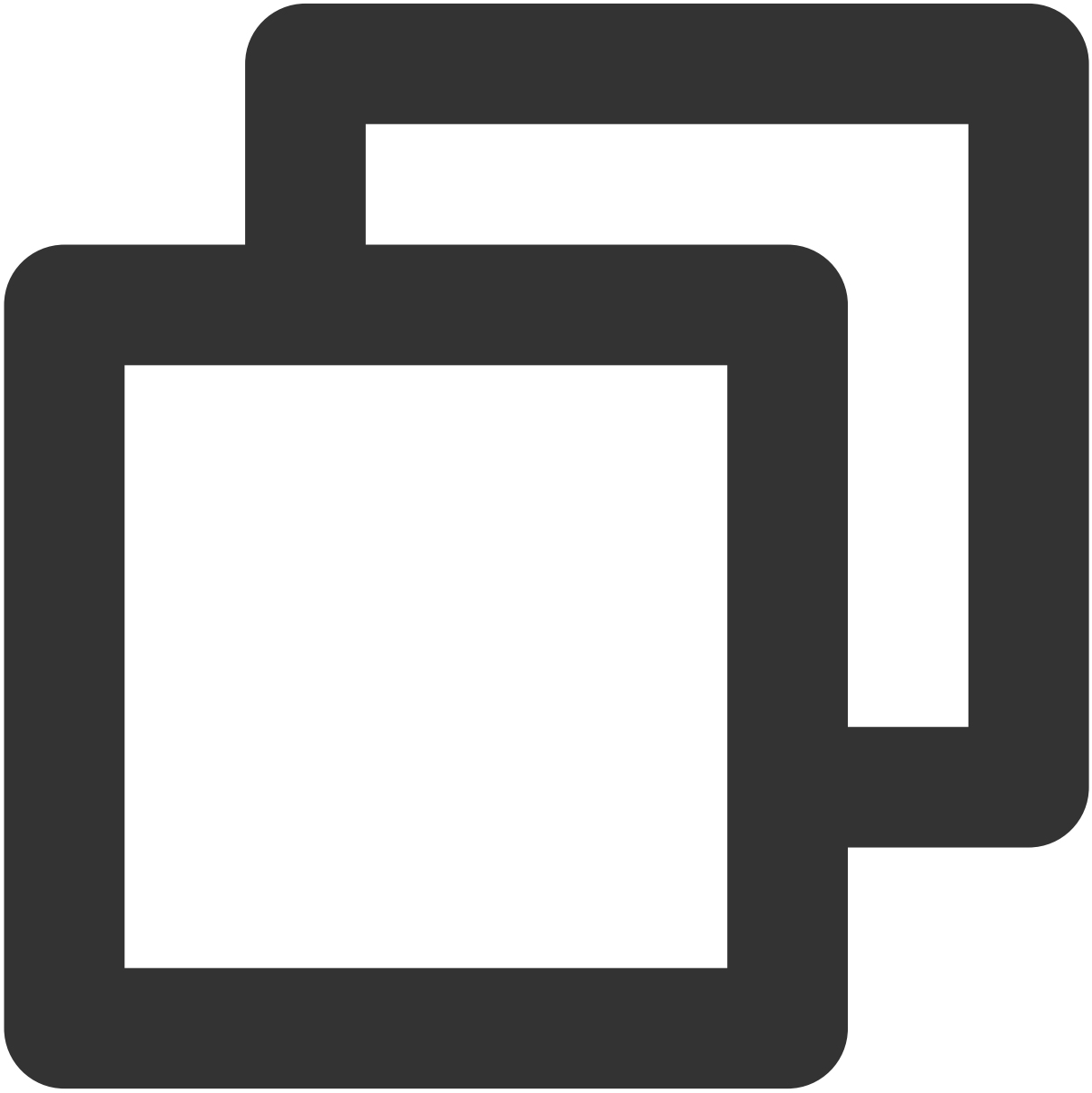
2. 素材のAvatarソースデータを取得します。



```
@implementation AvatarViewController
_resManager = [[AvatarResManager alloc] init];
NSDictionary *avatarDict = self.resManager.getMaleAvatarData;
@end


@implementation AvatarResManager

- (NSDictionary *)getMaleAvatarData
{
    if (!_maleAvatarDict) {
```

```
            NSString *resDir = [self avatarResPath:AvatarGenderFemale];
            NSString *savedConfig = [self getSavedAvatarConfigs:AvatarGenderMale];
            // sdkインターフェースで素材ソースデータを解析します
            _maleAvatarDict = [XMagic getAvatarConfig:resDir exportedAvatar:savedCo
    }
    return _maleAvatarDict;
}
@end
```

3. アバター制作操作を行います。

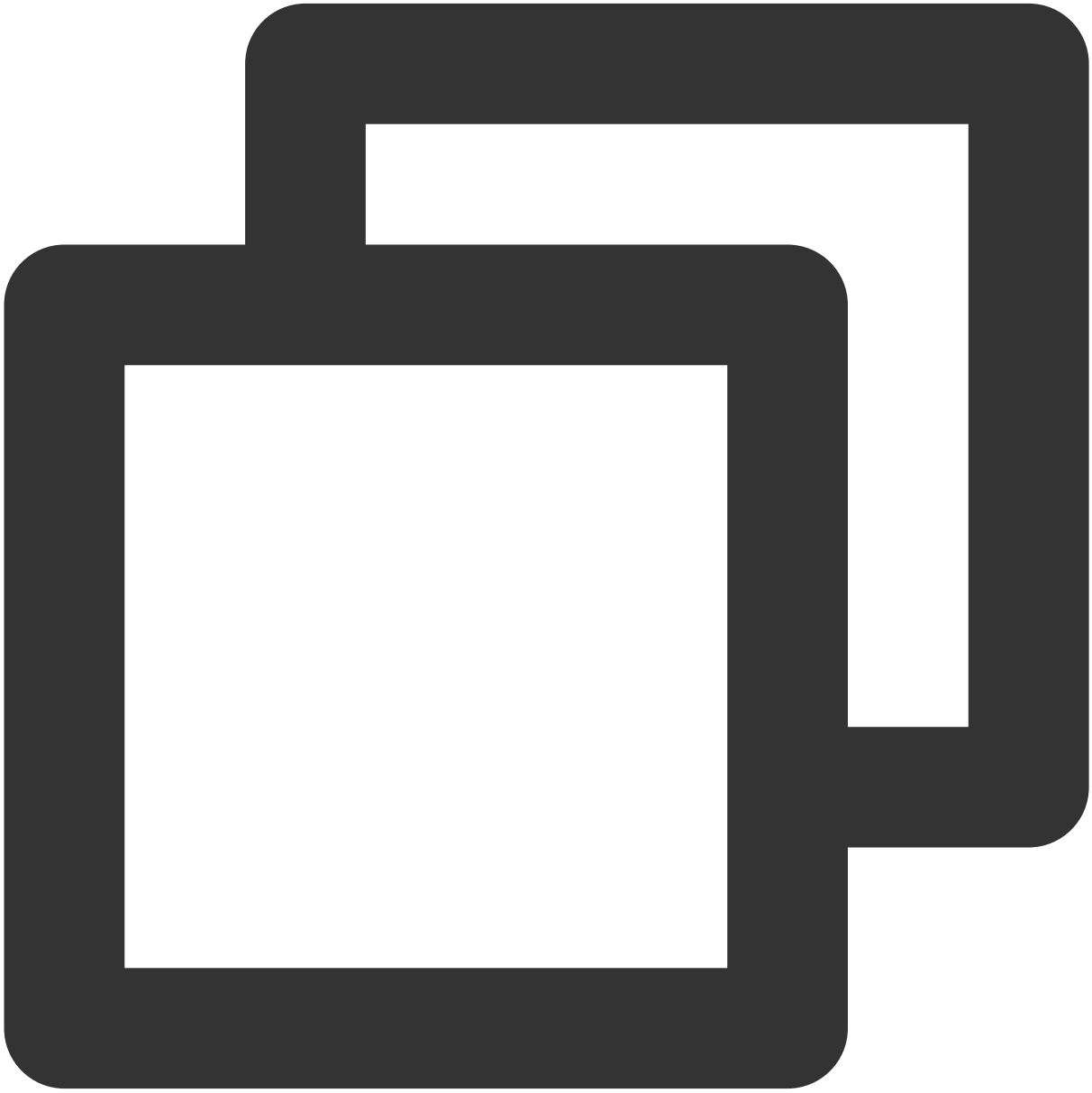


```
// sdkインターフェースで解析した素材ソースデータから必要なキャラクターのavatarオブジェクトを取得し
```

```
NSMutableArray *avatars = [NSMutableArray array];
// avatarConfigはsdkインターフェースgetAvatarConfig:exportedAvatar:から取得したavatarオブ
[avatars addObject:avatarConfig];
// アバター制作/着せ替えインターフェース。呼び出すと現在の素材で表現されるキャラクターをリアルタイム
[self.beautyKit updateAvatar:avatars];
```

4. アバター制作文字列のエクスポート：

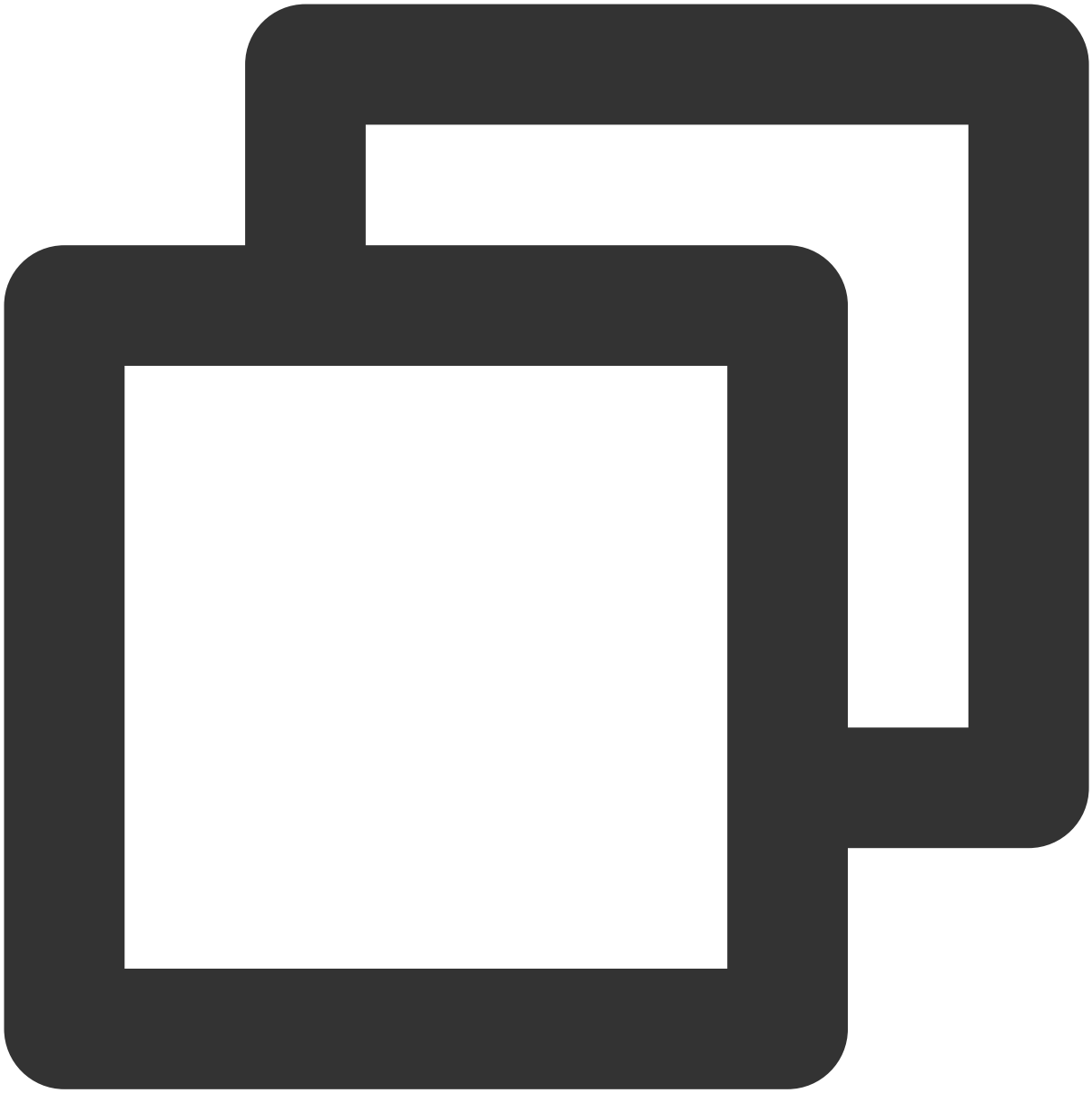
現在のAvatar設定のオブジェクトを文字列としてエクスポートし、カスタム保存できます。



```
- (BOOL)saveSelectedAvatarConfigs:(AvatarGender)gender
{
```

```
    NSMutableArray *avatarArr = [NSMutableArray array];
    NSDictionary *avatarDict = gender == AvatarGenderMale ? _maleAvatarDict : _fema
    // 1、選択したavatarオブジェクトをトラバースして探します
    for (NSArray *arr in avatarDict.allValues) {
        for (AvatarData *config in arr) {
            if (config.type == AvatarDataTypeSelector) {
                if (config.isSelected) {
                    [avatarArr addObject:config];
                }
            } else {
                [avatarArr addObject:config];
            }
        }
    }
    // 2、sdkインターフェースを呼び出し、選択したavatarオブジェクトを文字列としてエクスポートしま
    NSString *savedConfig = [XMagic exportAvatar:avatarArr.copy];
    if (savedConfig.length <= 0) {
        return NO;
    }
    NSError *error;
    NSString *fileName = [self getSaveNameWithGender:gender];
    NSString *savePath = [_saveDir stringByAppendingPathComponent:fileName];
    // ディレクトリが存在するかどうかを判断し、存在しない場合はディレクトリを作成します
    BOOL isDir;
    if (![[NSFileManager defaultManager] fileExistsAtPath:_saveDir isDirectory:&isD
        [[NSFileManager defaultManager] createDirectoryAtPath:_saveDir withInte
    }
    // 3、エクスポートした文字列をサンドボックスに書き込み、次回取り出して使用できるようにします
    [savedConfig writeToFile:savePath atomically:YES encoding:NSUTF8StringEncoding
    if (error) {
        return NO;
    }
    return YES;
}
```

5. バーチャル背景とリアル背景を切り替えます。



```
- (void)bgExchangeClick:(UIButton *)btn
{
    btn.selected = !btn.isSelected;
    NSDictionary *avatarDict = self.resManager.getFemaleAvatarData;
    NSArray *array = avatarDict[@"background_plane"];
    AvatarData *selConfig;
    // 背景も実際にはavatarオブジェクトの1つです。categoryはbackground_planeであり、背景を置
    for (AvatarData *config in array) {
            if ([config.Id isEqual:@"none"]) {
                    if (btn.selected) {
                            selConfig = config;
```

```
                            break;
                    }
            } else {
                    selConfig = config;
            }
    }
    [self.beautyKit updateAvatar:@[selConfig]];
}
```

Avatar SDKの説明

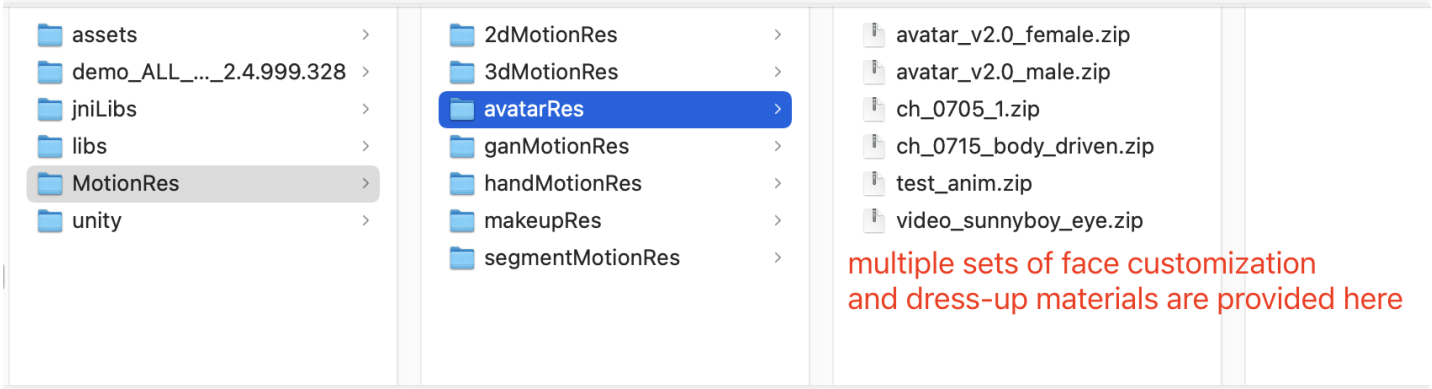
最終更新日：：2022-12-22 17:44:25

SDKへのアクセス

SDKのダウンロード、接続、認証、Demoクイックスタートについては、Tencent Effectの独立した統合をご参照ください。

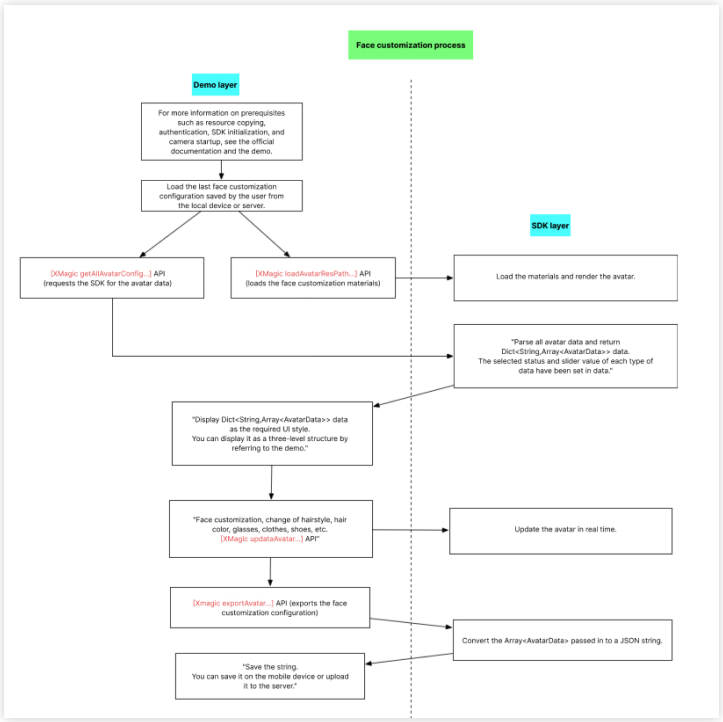
アバター制作素材の準備

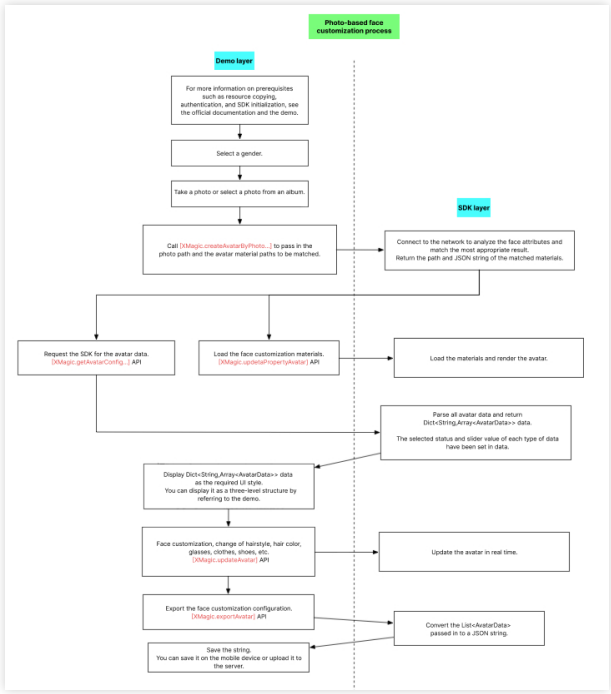
現在Tencentでは、SDKと共にいくつかのアバター制作、着せ替え素材をご提供しています。素材はSDKの解凍後の `MotionRes/avatarRes` ディレクトリ内にあります。他の動的エフェクト素材と同様に、プロジェクトのassetsディレクトリにcopyする必要があります



アバター制作フローとSDKインターフェース

アバター制作フロー	写真撮影によるアバター制作のフロー





XMagicApiのロードデータ、アバター制作、エクスポート設定、写真撮影アバター制作インターフェースの詳細は次のとおりです。

1. Avatarソースデータ取得インターフェース（getAvatarConfig）

```
+ (NSDictionary <NSString *, NSArray *>* _Nullable)getAvatarConfig:(NSString * _Nullable)resPath exportedAvatar:(NSString *_Nullable)exportedAvatar;
```

- **入力パラメータ**：
  - resPath：Avatar素材のスマートフォン上の絶対パスです。
    例： `/var/mobile/Containers/Data/Application/C82F1F7A-01A1-4404-8CF6-131B26B4DA1A/Library/Caches/avatarMotionRes.bundle/avatar_v2.0_male` 。
  - exportedAvatar：ユーザーが前回のアバター制作後に保存したデータであり、JSON形式の文字列です。初めて使用する場合またはユーザーがそれまでに保存したことがない場合、この値はnilとなります。
- **出力パラメータ**：
  NSDictionaryの形式で返します。dictionaryのkeyはデータのcategoryです。詳細についてはTEDefineクラスをご覧ください。dictionaryのvalueはこのcategory下のすべてのデータです。アプリケーション層でこのdictionaryを取得した後、必要に応じて希望のUIスタイルを表示します。

2. Avatar素材ロードインターフェース（loadAvatar）

```
- (void)loadAvatar:(NSString * _Nullable)resPath exportedAvatar:(NSString * _Null
able)exportedAvatar;
```

- **入力パラメータ**：
  - resPath：avatar素材のスマートフォン上の絶対パスです。
    例： `/var/mobile/Containers/Data/Application/C82F1F7A-01A1-4404-8CF6-131B26B4DA1A/Library/Caches/avatarMotionRes.bundle/avatar_v2.0_male` 。
  - exportedAvatar：ユーザーが前回のアバター制作後に保存したデータであり、json形式の文字列です。初めて使用する場合またはユーザーがそれまでに保存したことがない場合、この値はnilとなります。
- **出力パラメータ**：
  NSDictionaryの形式で返します。dictionaryのkeyはデータのcategoryです。詳細についてはTEDefineクラスをご覧ください。dictionaryのvalueはこのcategory下のすべてのデータです。アプリケーション層でこのdictionaryを取得した後、必要に応じて希望のUIスタイルを表示します。

3. アバター制作、着せ替えインターフェース（updateAvatar）

```
- (void)updateAvatar:(NSArray<AvatarData *> *_Nonnull)avatarDataList;
```

呼び出すと現在の素材のプレビューキャラクターをリアルタイムに更新します。1つのAvatarDataオブジェクトは1つのアトミック設定であり（髪型変更など），一度に複数のアトミック設定を渡すことができます（髪型と髪色の両方を変更するなど）。このインターフェースは渡されるAvatarDataの有効性をチェックし、有効なものはSDKに設定し、無効なデータにはcallbackを返します。

- 例えば髪型の変更を要求しているのに、髪のモデルファイル（AvatarDataのvalueフィールド内に設定）がローカルに見つからない場合、このAvatarDataは無効と判断されます。
- また例えば、瞳マップの変更を要求しているのに、マップファイル（AvatarDataのvalueフィールド内に設定）がローカルに見つからない場合、このAvatarDataは無効と判断されます。

4. アバター制作設定エクスポートインターフェース（exportAvatar）

```
+ (NSString *_Nullable)exportAvatar:(NSArray <AvatarData *>*_Nullable)avatarDataL
ist;
```

ユーザーがアバターを制作する際、AvatarDataのselectedステータスまたはブレンドシェイプ値を変更する場合があります。制作完了後、新しいフルAvatarDataリストを渡すとJSON文字列をエクスポートできます。この文字列はローカルに保存することも、サーバーにアップロードすることもできます。
このエクスポートされた文字列には2つの用途があります。

- 次回再びXMagicの**loadAvatar**インターフェースでこのAvatar素材をダウンロードする際、このJSON文字列を**exportedAvatar**として設定すると、ユーザーが前回作成したキャラクターをプレビューで表示することができます。

- 上記のように、getAllAvatarDataを呼び出す際にこのパラメータを渡すことで、Avatarソースデータ内の選択状態およびブレンドシェイプ値を変更できるようにする必要があります。

5. 写真撮影アバター制作インターフェース（createAvatarByPhoto）

このインターフェースにはネットワーク接続が必要です。

```
+ (void)createAvatarByPhoto:(NSString * _Nullable)photoPath avatarResPaths:(NSArray <NSString *>* _Nullable)avatarResPaths isMale:(BOOL)isMale success:(nullable void (^)(NSString *_Nullable matchedResPath, NSString *_Nullable srcData))success failure:(nullable void (^)(NSInteger code, NSString *_Nullable msg))failure;
```

- **photoPath**：写真のパスです。顔が確実に画面の中央に位置するようにしてください。画面内に含まれる顔は1つだけにすることをお勧めします。複数の顔がある場合、SDKはその中の1つを選択します。写真の短辺は500px以上を推奨します。これより小さいと認識効果に影響する可能性があります。

- **avatarResPaths**：複数のAvatar素材セットを渡すことができます。SDKは写真の分析結果に基づいて、最も適した素材のセットを選択し、自動的にアバター制作を行います。

  > 注意：
  > 現在は1セットのみをサポートしています。複数のセットを渡した場合、SDKは最初の1セットのみを使用します。

- **isMale**：男性かどうか。この属性は現在使用されていませんが、今後SDKが最適化される場合を考え、正確に渡すことをお勧めします。

- **success**：成功のコールバックです。 matchedResPathはマッチした素材パス、srcDataはマッチ結果です。上記で述べたexportAvatarインターフェースが返すものは同じ意味となります。

- **failure**：失敗のコールバックです。codeはエラーコード、msgはエラーメッセージです。

6. ダウンロードした設定ファイルを対応するフォルダ内に配置する（addAvatarResource）

```
+ (void)addAvatarResource:(NSString * _Nullable)rootPath category:(NSString * _Nullable)category filePath:(NSString * _Nullable)filePath completion:(nullable void (^)(NSError * _Nullable error, NSArray <AvatarData *>* _Nullable avatarList))completion;
```

このインターフェースは主にAvatarパーツの動的ダウンロードのシナリオに用いられます。例としては、Avatar素材内に10種類の髪型があり、その後1種類の髪型をクライアントに動的に送信したい場合、ダウンロード完了後に圧縮パッケージを取得してからこのインターフェースを呼び出し、圧縮パッケージのパスをSDKに渡します。SDKはこの圧縮パッケージを解析し、対応するcategoryディレクトリ下に保存します。getAllAvatarDataインターフェースを次に呼び出す際、SDKは新たに追加されたこのデータを解析することができます。

パラメータの説明：

- **rootPath**：Avatar素材のルートディレクトリです。例えば `/var/mobile/Containers/Data/Application/C82F1F7A-01A1-4404-8CF6-131B26B4DA1A/Library/Caches/avatarMotionRes.bundle/avatar_v2.0_male` などです。
- **category**：ダウンロードしたこの設定のカテゴリーです。
- **zipFilePath**：ダウンロードしたZIPパッケージを保存するローカルアドレスです。
- **completion**: 結果のコールバックです。errorはエラーメッセージ、avatarListは解析したavatarデータ配列を表します。

7. カスタムイベントの送信

カスタムイベントを送信します。例：顔がない場合のアイドル状態の表示の有効化。

```
- (void)sendCustomEvent:(NSString * _Nullable)eventKey eventValue:(NSString * _Nullable)eventValue;
```

- **eventKey**：カスタムイベントのkeyです。TEDefineのAvatarCustomEventKeyを参照できます。
- **eventValue**：カスタムイベントのvalueです。 JSON文字列であり、例えば `@{@"enabel" : @(YES)}` をjson文字列に変換するか、または`@"{\\"enable\\" : true}"` を直接書き込みます。

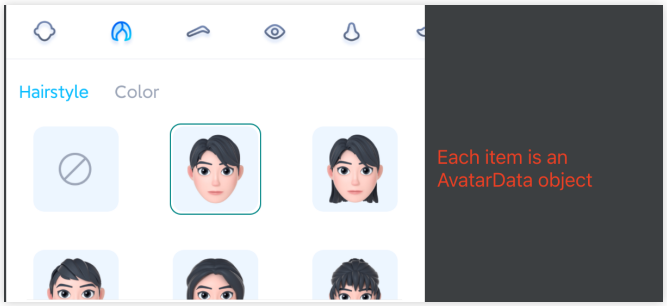
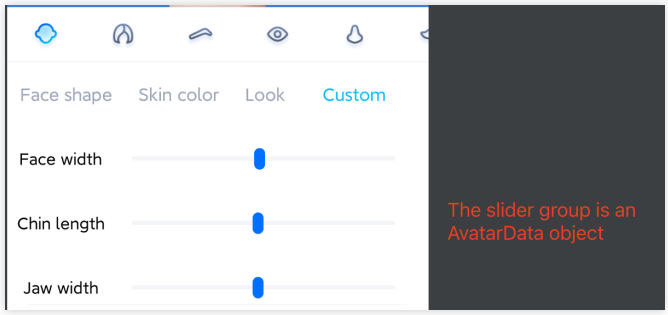
8. AvatarDataの呼び出し

これらのインターフェースのコアはすべてAvatarDataクラスです。その主な内容は次のとおりです。

```
/// @brief アバター制作設定タイプ
@interface AvatarData : NSObject
/// 例えば、輪郭、目の微調整などです。TEDefineでは標準のcategoryを定義し、それでニーズを満たせない場合はcategory文字列をカスタマイズすることもできます。既存のものと競合しなければそれでよく、空にすることはできません
@property (nonatomic, copy) NSString * _Nonnull category;
// 具体的な各itemまたは各微調整項目を識別します。例えば各眼鏡にはすべて個々のidがあり、各微調整項目にもすべて個々のidがあります。空にすることはできません
@property (nonatomic, copy) NSString * _Nonnull Id;
// selectorタイプまたはAvatarDataTypeSlider値タイプ
@property (nonatomic, assign) AvatarDataType type;
/// selectorタイプの場合は、現在選択されているものの有無を表します
@property (nonatomic, assign) BOOL isSelected;
```

```
/// 顔、目、髪、トップス、靴などの体のある部位を制作します。値の設定方法については公式ドキュメントをご参照ください
@property (nonatomic, copy) NSString * _Nonnull entityName;
/// entityNameに対してどの操作(action)を実行するかを表します。 ルールについては公式ドキュメントをご参照ください
@property (nonatomic, copy) NSString * _Nonnull action;
/// entityNameに対して実行するactionの具体的な値を表します。 ルールについては公式ドキュメントをご参照ください
@property (nonatomic, copy) NSDictionary * _Nonnull value;
@end
```

1つのAvatarDataオブジェクトは1つのアトミック設定です（髪型変更、 頬の調整など）。

髪型変更	頬の調整
	

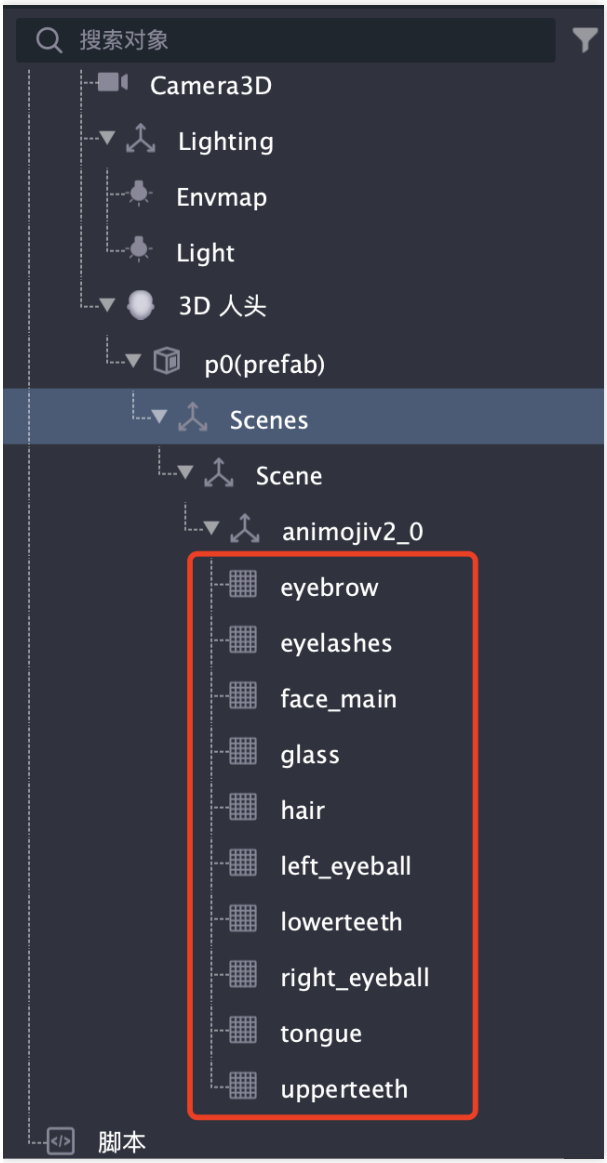
- アバター制作の際、データがselectorタイプの場合は、AvatarDataのselectedフィールドを変更します。例えばA、B、C、Dという4種類の眼鏡があり、Aをデフォルトで選択している場合、Aのselectedはtrue、B、C、Dはfalseとなります。ユーザーが眼鏡Bを選択した場合、それはBのselectedをtrueにし、A、C、Dをfalseにすることになります。
- アバター制作の際、データがsliderタイプの場合は、AvatarDataのvalueフィールドを変更します。valueフィールドはJsonObjectであり、その中にはいくつかのkey-valueペアがあります。key-valueの中のvalueをスライダーの値に変更するだけです。

AvatarDataの高度な説明（参考）

AvatarDataはSDKが素材ルートディレクトリのcustom_configsディレクトリから自動的に解析してアプリケーション層に返します。通常はAvataDataを手動で構築する必要はありません。
AvatarDataの中で、アバター制作において重要な役割を果たすものはentityName、action，valueの3つのフィールドです。これら3つのフィールドの値は、SDKが素材設定を解析する際に自動的に入力されます。多くの場合、この3つのフィールドの意味を理解する必要はありませんが、UI層で表示する際、スライダータイプの場合は、value内のブレンドシェイプkey-valueを解析し、UIの操作と対応させる必要があります。

entityNameフィールド

アバター制作の際は、どの部位（顔、目、髪、トップス、靴など）を制作するかを明確に指定する必要があります。entityNameフィールドはこれらの体の部位の名前を記述するためのものです。TencentのDemo素材の場合、TencentEffectStudioで素材プロジェクト（xxx.studio）を開くと、このようなリストが表示されます。



リスト内の各項目がそれぞれ体の部位に対応しています。**命名する際は、その命名がstudioプロジェクトにおいてグローバルで一意であることを確実にする必要があります。**

actionフィールド

actionフィールドはentityNameに対してどの操作（action）を実行するかを表します。SDK内では5種類のactionを定義しています。各actionの意味およびvalueの要件は次のとおりです。

action	意味	value要件

action	意味	value要件
changeColor	現在のマテリアルの色を変更します。基本色、放射光色などの色属性が含まれます	JsonObjectタイプ。入力必須です。下記参照
changeTexture	現在のマテリアルのマップを変更します。色のテクスチャマップ、金属の粗さのテクスチャマップ、AOテクスチャマップ、法線テクスチャマップ、放射光テクスチャマップなどが含まれます	JsonObjectタイプ。入力必須です。下記参照
shapeValue	blendShape値を変更します。通常、顔の細部のブレンドシェイプの微調整に用いられます	JsonObjectタイプ。このうちkeyはブレンドシェイプ名、valueはfloatタイプの値です。入力必須です。下記参照
replace	サブモデルの置き換え（眼鏡、髪型、衣服などの置き換えなど）	JsonObjectタイプ。内部には新しいサブモデルの3D変換情報、モデルパス、マテリアルパスを記述しています。現在の位置にあるサブモデルを非表示にしたい場合はnullを使用します。下記参照
basicTransform	位置、回転、ズームを調整します。通常はカメラの遠近、角度の調整に用いることで、モデルの全身と半身アングルの切り替えを実現します	JsonObjectタイプ。入力必須です。下記参照

valueフィールド

- actionがchangeColorであるシナリオ
- actionがchangeTextureであるシナリオ
- actionがshapeValueであるシナリオ
- actionがreplaceであるシナリオ
- actionがbasicTransformであるシナリオ

設定例：

```
{
"key": "baseColorFactor",
"value": [43, 26, 23, 255],
"subMeshIndex":0,
"materialIndex":0
}
```

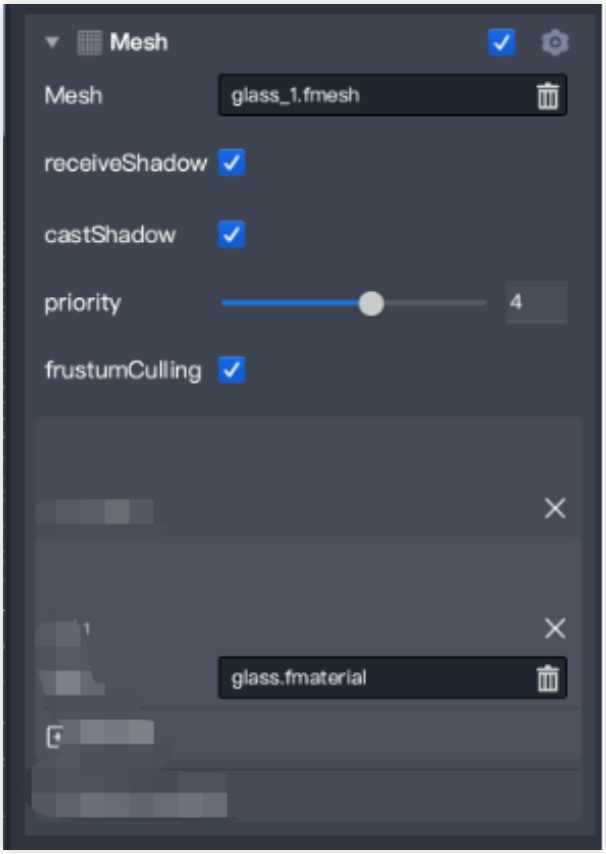
- **key**：入力必須です。現在のマテリアルの色を変更します。基本色、放射光色、その他のカスタム色などの色属性が含まれます。

keyが具体的にどの値をとるかは、現在そのマテリアルを使用しているshaderに関連します。上記のスクリーンキャプチャで使用しているのは内蔵のlit_fadeというshaderであり、そこで定義する基本色の名前はbaseColorFactor、放射光色はemissiveFactorであるため、keyの値はそのどちらかとします。また、マテリアルファイルをテキスト形式で直接開くことも、その中で各色のkey値を確認することもできます。例えばここで使用しているのは `hair.fmaterial` というマテリアルファイルであり、TEStudioプロジェクトディレクトリで見つけて開きます。その内容は次のとおりです。

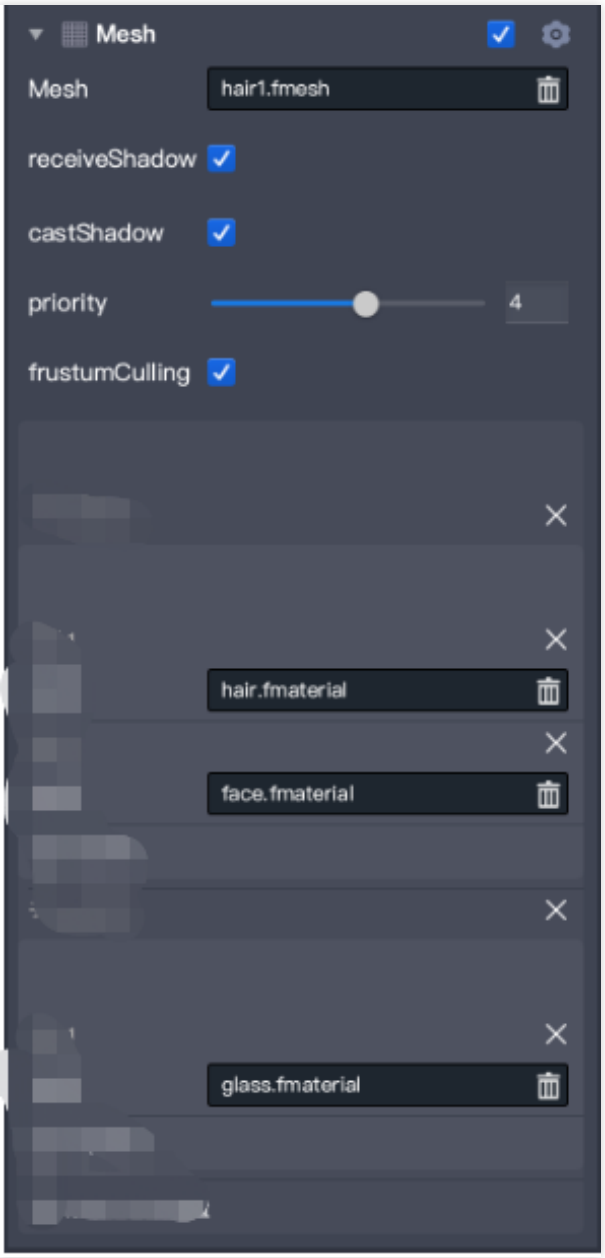
hair.fmaterial


```
{
    "renderState": {
        "colorWrite": true,
        "depthWrite": true,
        "depthCulling": true,
        "doubleSided": true,
        "cullingMode": "BACK"
    },
    "shaderResourceKey": "internal:lit_fade",
    "values": {
        "baseColorFactor": [
            0.6745098039215687,
            0.5607843137254902,
            0.36470588235294116,
            1
        ],
        "baseColorEnableTexture": false,
        "baseColorMap": "",
        "baseColorIndex": 0,
        "baseColorTexturePremultiplied": false,
        "metallicFactor": 0,
        "roughnessFactor": 0.66,
        "metallicRoughnessEnableTexture": false,
        "metallicRoughnessMap": "",
        "metallicRoughnessIndex": 0,
        "metallicSamplingChannel": 2,
        "roughnessSamplingChannel": 1,
        "aoEnableTexture": false,
        "occlusionMap": "",
        "aoIndex": 0,
        "aoSamplingChannel": 0,
        "aoStrength": 1,
        "normalEnableTexture": false,
        "normalMap": "",
        "normalIndex": 0,
        "normalScale": 1,
        "normalEnableGReverse": false,
        "emissiveFactor": [
            0,
            0,
            0,
            1
        ],
        "emissiveEnableTexture": false,
        "emissiveMap": "",
        "emissiveIndex": 0,
        "emissiveStrength": 1,
        "emissiveEnableTextureColorMultiply": true
    }
}
```

- **value**：入力必須です。4つの値はそれぞれRGBAです。
- **subMeshIndexおよびmaterialIndex**：任意入力です。このフィールドに入力しない場合、SDKではデフォルト値の0をとります。
  TEStudioでは、Mesh(1つのサブモデルであると理解することができます。例：眼鏡、髪、顔)には一般的にサブMeshは設定されておらず、なおかつ1つのMeshには1つのマテリアルしか存在しません。下図をご覧ください。



一部の比較的複雑なMeshには複数のサブMeshが存在する場合があり、1つのMeshが複数のマテリアルを持つこともあります。下図をご覧ください。



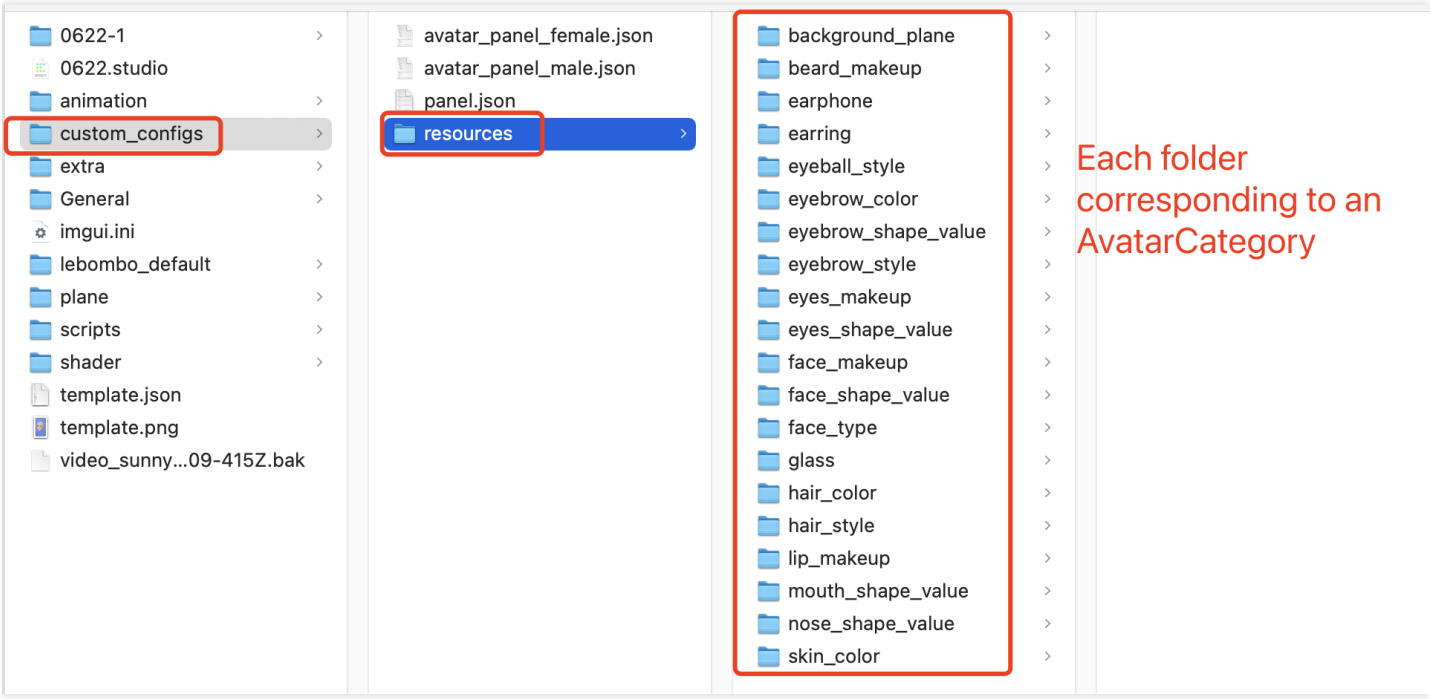
このため、changeColorを行いたい場合は、どのマテリアルのcolorをchangeしたいのかを明確に指示する必要があります。subMeshIndexとmaterialIndexは具体的なマテリアル位置の特定に用います。この2つのindexの値を決定するには、素材プロジェクトディレクトリ下のtemplate.jsonを開き、現在のmeshを見つけ、subMeshConfigs属性を確認する必要があります。例えば“hair”を検索するとhairの設定の位置を特定でき、次のようなsubMesh設定を確認することができます。

```
"frustumCulling": true,
"level": 290,
"meshResourceKey": "854f65c1-75c5-4a8d-8a4b-6f3f90803b03",
"needReload": true,
"paused": false,
"priority": 4,
"receiveShadow": true,
"skinInfoResourceKey": "",
"subMeshConfigs": [
   {
      "materialResourceKeys": [
         "6224dce3-6606-4903-b5fd-3b7b7cf21cb3",
         "70fb91ad-5db9-4599-a009-b8ad530b5df5"
      ]
   },
   {
      "materialResourceKeys": [
         "64a8353e-537a-41ec-b85e-3063cdd78f3a"
      ]
   }
],
"type": "MeshRenderer3DComponentV3",
"version": 30
```


その後、subMeshIndexとmaterialIndexを組み合わせることで、具体的なマテリアルの位置を特定することができます。

Avatar制作・着せ替えデータの設定

Avatar属性設定はresourcesフォルダ下に配置します（パス： `素材/custom_configs/resources` ）。



これらの設定ファイルは自動生成されるため、通常は手動で設定する必要はありません。自動生成の方法は次のとおりです。

設計者はTencentEffectStudioを使用してキャラクターのセットを設計後、Tencentの提供するresource_generator_guiというapp（現在はMacOSプラットフォームのみサポートしています）を実行するだけでこの設定を自動生成できます。詳細については設計仕様をご参照ください。

Android
Avatarクイックアクセス

最終更新日：：2023-02-27 14:18:15

AvatarはTencent Effectの機能の一部であるため、接続の際はEffectのアクセスドキュメントを参照した上でAvatar素材をロードする必要があります。Effectを接続済みの場合は、下記のステップ1の内容を無視してください。
1. Tencent Effectのドキュメントに従って接続します。Tencent Effectの独立した統合をご参照ください。
2. 次の方法に従ってAvatar素材をロードします。

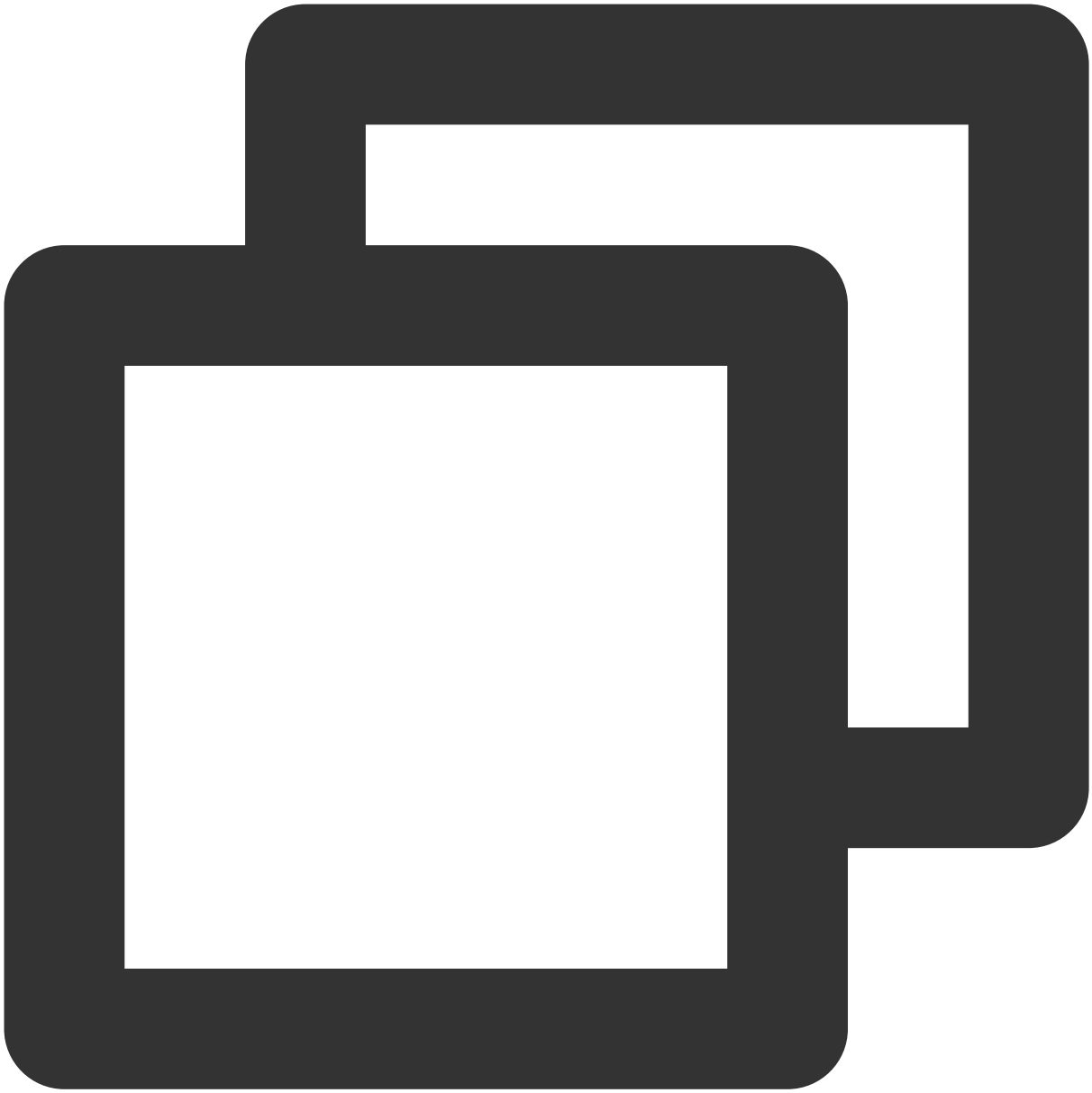
Avatar機能を使用するための具体的な手順

ステップ1：Demo内のファイルのコピー

1. 対応するdemoプロジェクトを公式サイトからダウンロードし、解凍します。
2. Demo内の `demo/app/assets/MotionRes/avatarRes` フォルダをご自身のプロジェクトにコピーします（Demoと同じ位置にします）。
3. Demo内の `com.tencent.demo.avatar` フォルダ下のすべてのクラスをプロジェクトにコピーします。

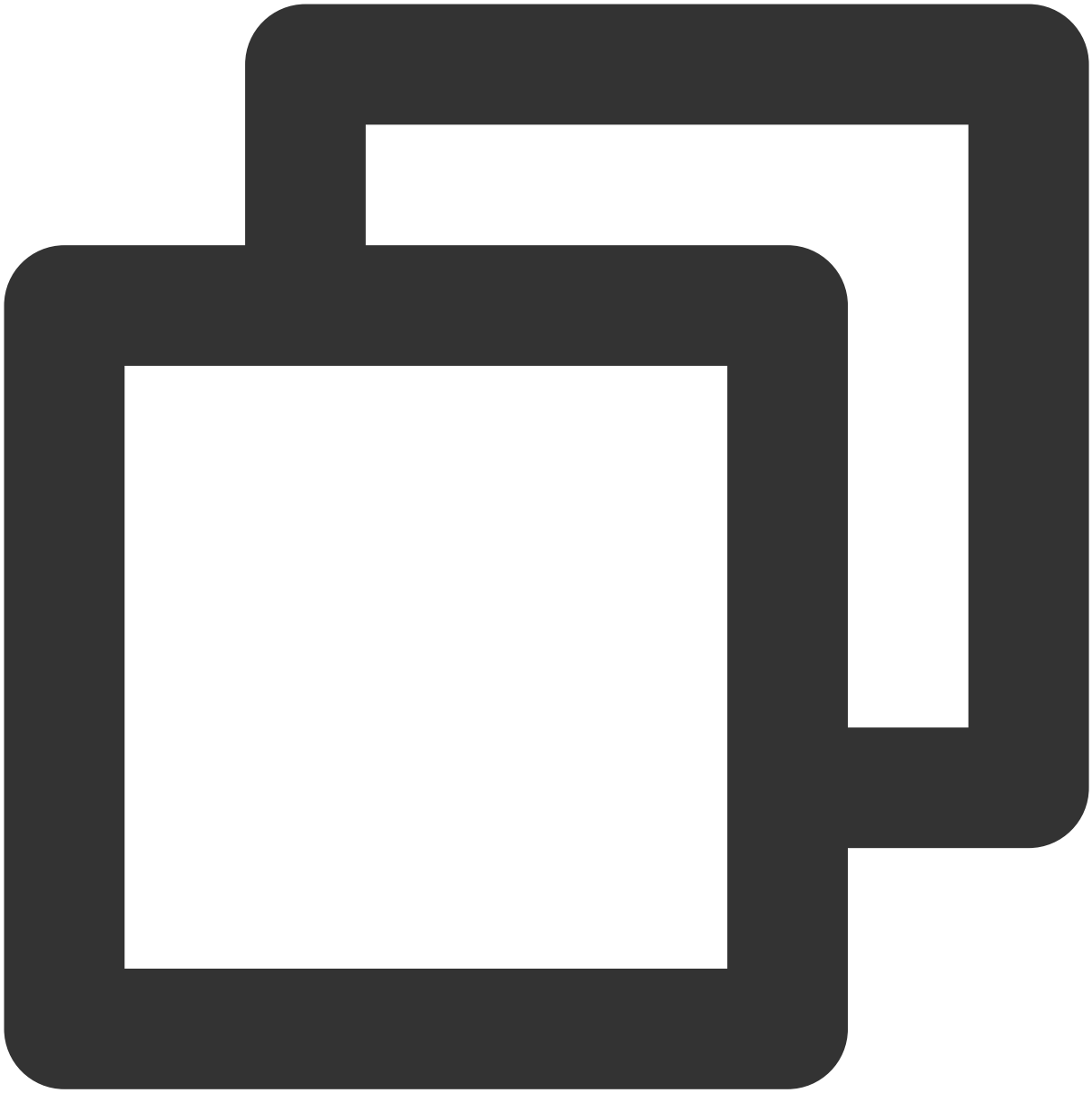
ステップ2：重要コードの追加

Demo内の `com.tencent.demo.avatar.AvatarActivity` クラスを参照し、次のコードを追加します
1. ページのxmlファイル内でパネル情報を設定します。



```
<com.tencent.demo.avatar.view.AvatarPanel
        android:id="@+id/avatar_panel"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        app:layout_constraintBottom_toBottomOf="parent" />
```

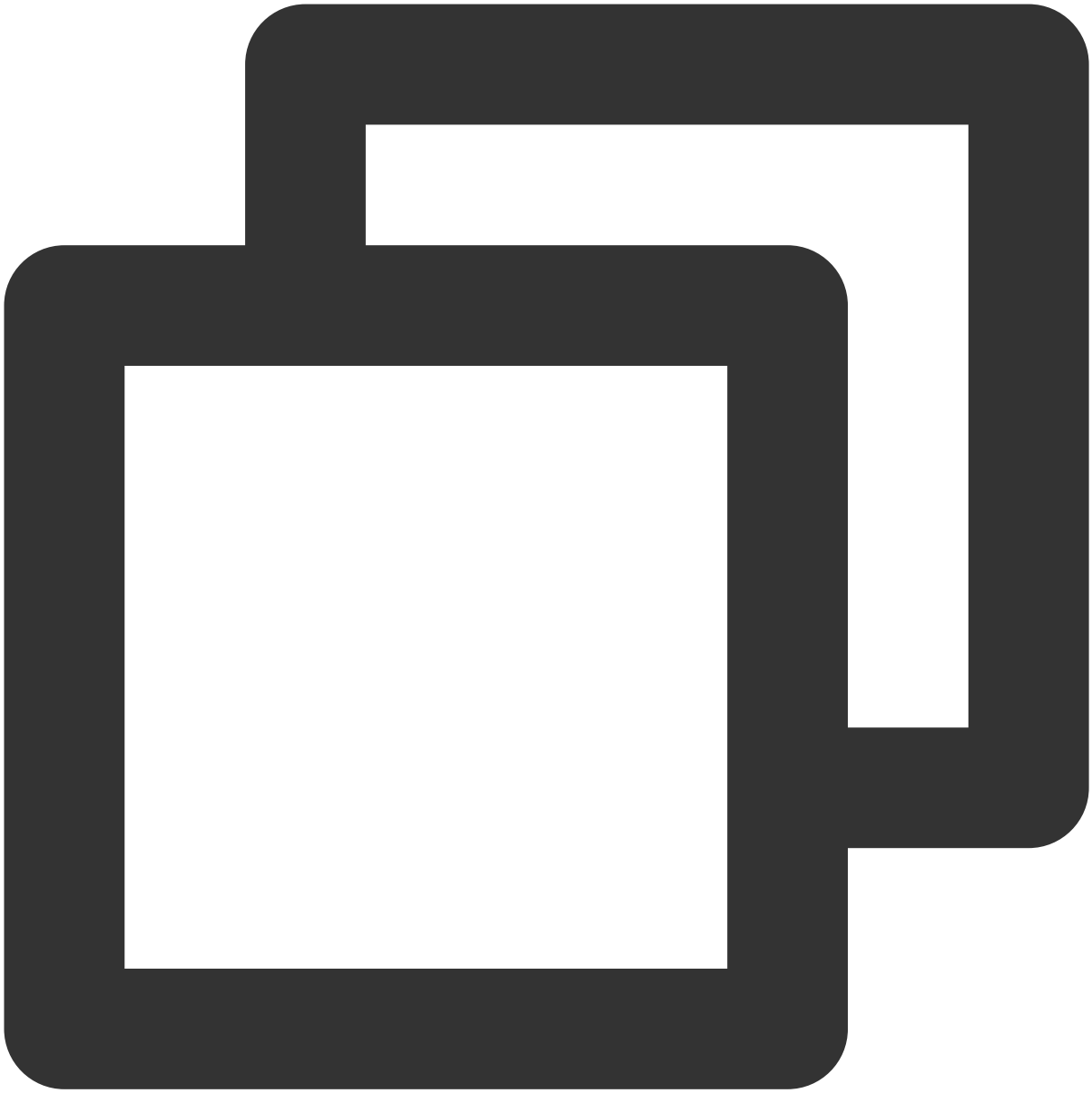
2. ページ内でパネルオブジェクトを取得し、対応するデータコールバックインターフェースを設定します。



```
avatarPanel.setAvatarPanelCallBack(new AvatarPanelCallBack() {
            @Override
            public void onItemChecked(MainTab mainTab, AvatarItem avatarItem) {
                if (avatarItem.avatarData == null && URLUtil.isNetworkUrl(avatarIte
                    downloadAvatarData(avatarItem, () -> updateConfig(avatarItem));
                } else {
                    updateConfig(avatarItem);
                    List<AvatarData> bindAvatarData = AvatarResManager.getAvatarDat
                    mXmagicApi.updateAvatar(bindAvatarData, AvatarActivity.this);
                }
            }
```

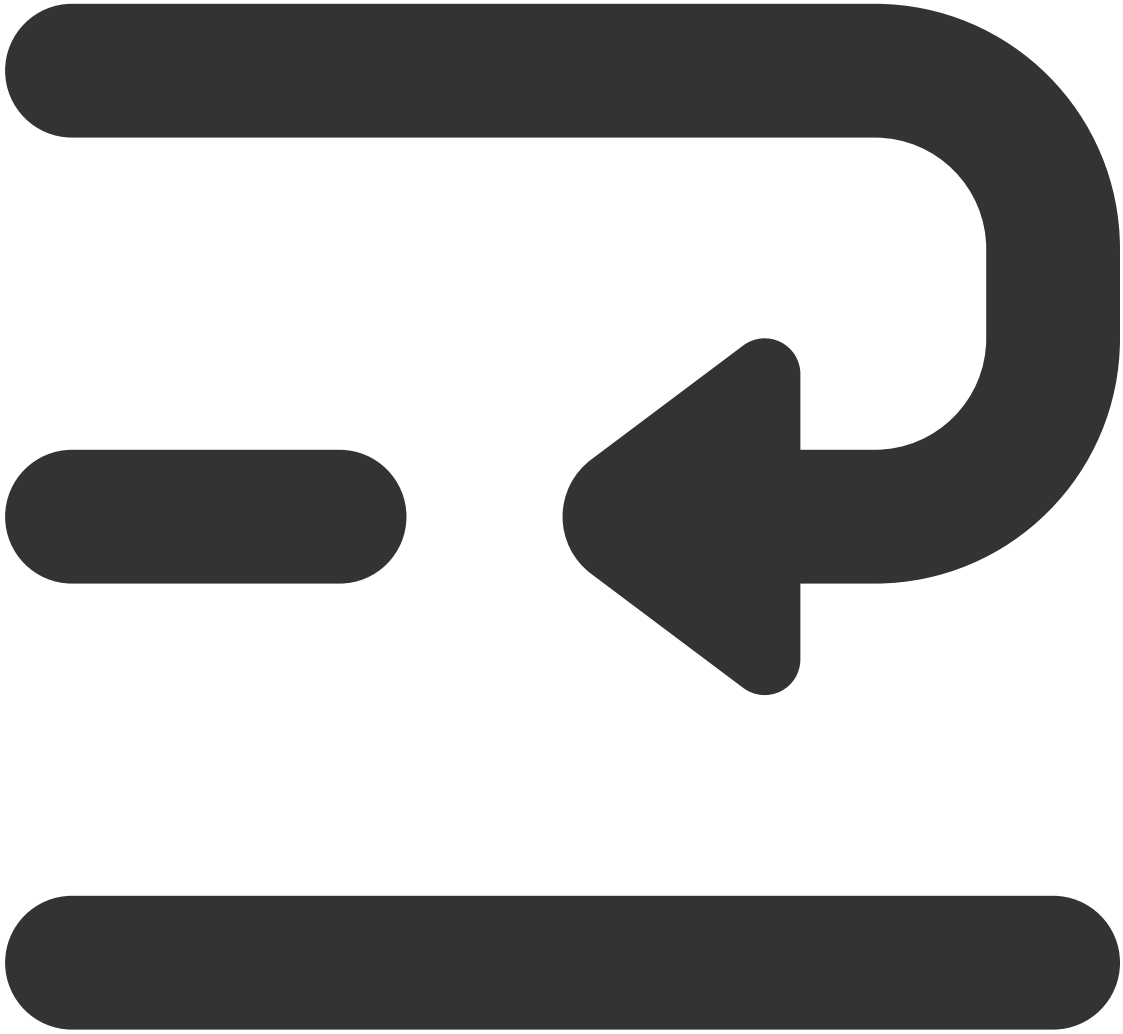
```
            @Override
            public void onItemValueChange(AvatarItem avatarItem) {
                updateConfig(avatarItem);
            }

            @Override
            public boolean onShowPage(AvatarPageInf avatarPageInf, SubTab subTab) {
                if (subTab != null && subTab.items != null && subTab.items.size() >
                    AvatarItem avatarItem = subTab.items.get(0);
                    if (avatarItem.type == AvatarData.TYPE_SLIDER && avatarItem.ava
                        downloadAvatarData(avatarItem, () -> {
                            if (avatarPageInf != null) {
                                avatarPageInf.refresh();
                            }
                        });
                        return false;
                    }
                }
                return true;
            }

            private void updateConfig(AvatarItem avatarItem) {
                if (mXmagicApi != null && avatarItem != null) {
                    List<AvatarData> avatarConfigList = new ArrayList<>();
                    avatarConfigList.add(avatarItem.avatarData);
                    mXmagicApi.updateAvatar(avatarConfigList, AvatarActivity.this);
                }
            }
        });
```

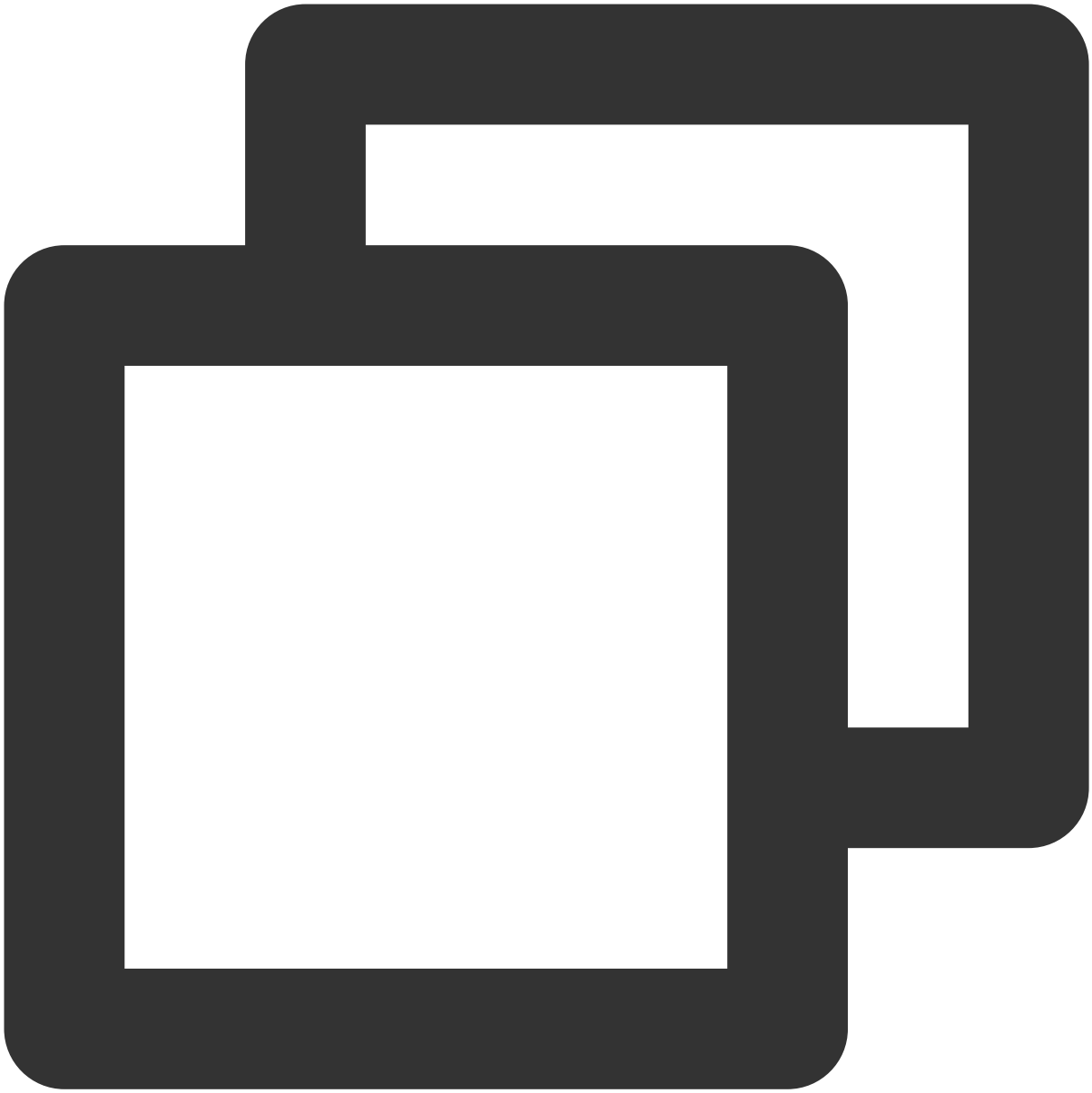
3. パネルデータを取得し、パネルに設定します。



```
AvatarResManager.getInstance().getAvatarData(avatarResName, getAvatarConfig(), allD
      avatarPanel.initView(allData);
 });
```

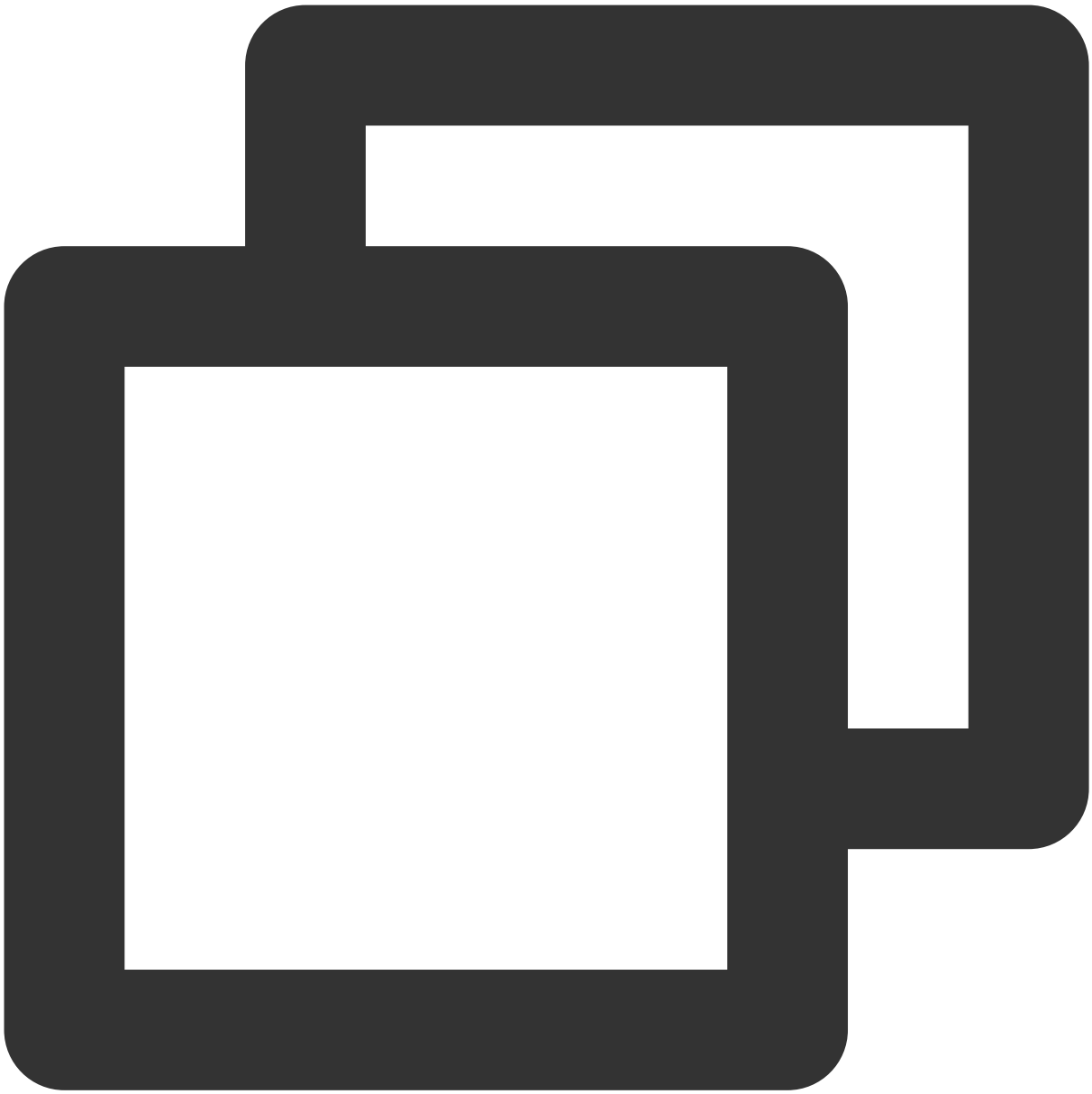
4. xmagicApiオブジェクトを作成し、アバター制作リソースをロードします。





```
private void initXMagic() {
    if (mXmagicApi == null) {
        mXmagicApi = XmagicApiUtils.createXmagicApi(getApplicationContext(),  n
        AvatarResManager.getInstance().loadAvatarRes(mXmagicApi, avatarResName,
        setCloseEye(true);//顔が検出されない場合は、スリープ/休止状態になります
        setAvatarPlaneType();
    } else {
        mXmagicApi.onResume();
    }
}
```

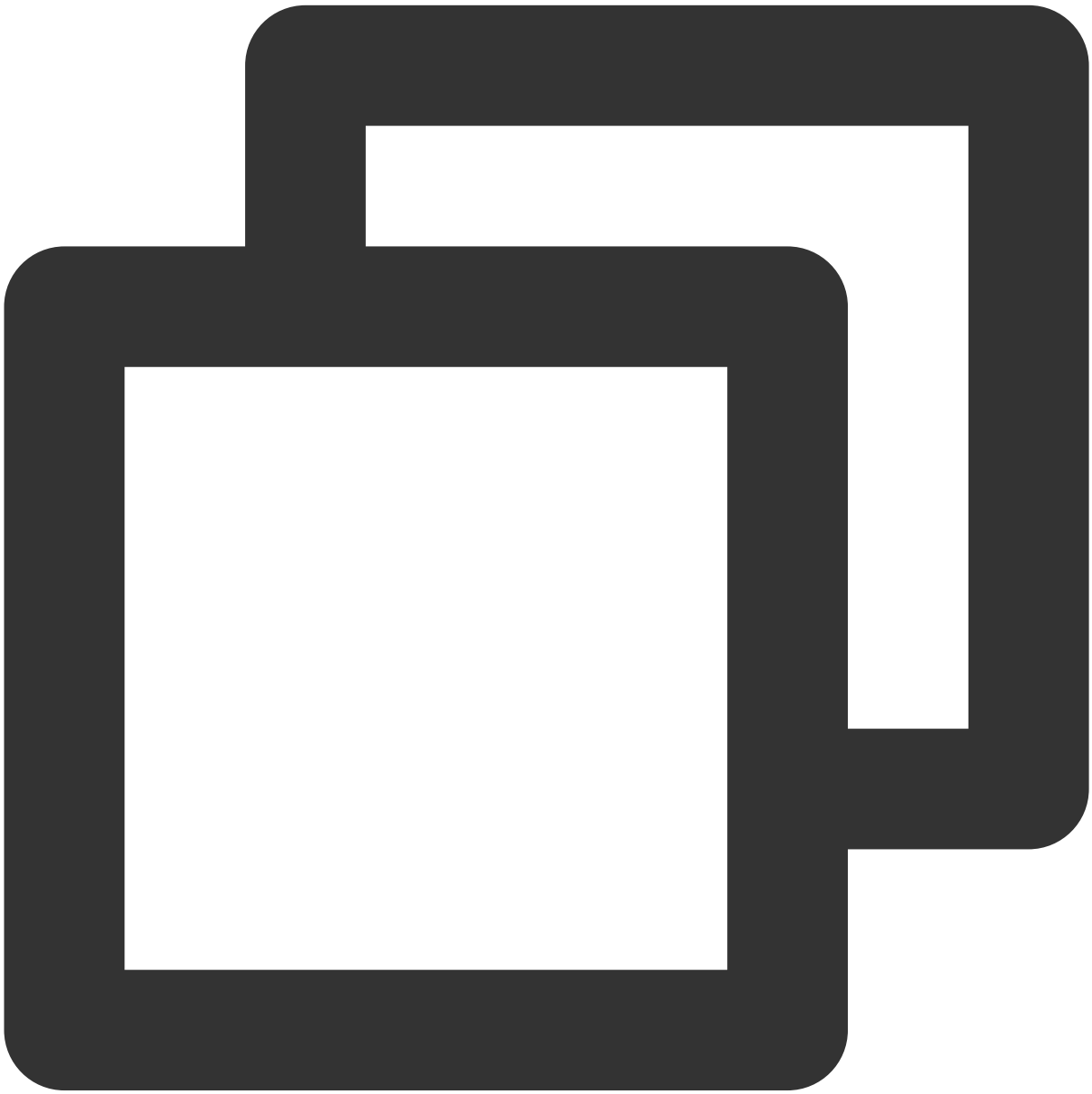
5. Avatar属性を保存します。Demo内のsaveAvatarConfigsメソッドを参照できます。



```
/**
     * ユーザーの設定した属性またはデフォルトの属性値を保存します
*/
private void saveAvatarConfigs() {
        if (mXmagicApi != null && avatarPanel != null) {
                List<AvatarData> avatarDataList = AvatarResManager.getUsedAvatarDat
                new Thread(() -> {
                        String content = XmagicApi.exportAvatarConfig(avatarDataLis
                        boolean result = FileUtil.writeContentToFile(AvatarResManag
```

```
                        String tip = result ? "save avatar data successfully " : "S
                        runOnUiThread(() -> Toast.makeText(getApplicationContext(),
                }).start();
        }
}
```

6. 背景の切り替えはDemo内のchangeAvatarBgTypeメソッドを参照します。



```
/**
 * 背景の切り替え
 */
private void changeAvatarBgType() {
```

```
        if (currentAvatarType == AvatarResManager.AvatarType.VIRTUAL_BG) {
                currentAvatarType = AvatarResManager.AvatarType.REAL_BG;
        } else {
                currentAvatarType = AvatarResManager.AvatarType.VIRTUAL_BG;
        }
        saveCurrentAvatarType();
        setAvatarPlaneType();
}

/**
 * モデル背景タイプの設定
 */
private void setAvatarPlaneType() {
        AvatarData avatarData = AvatarResManager
                .getInstance().getAvatarPlaneTypeConfig(avatarResName, currentAvata
        if (mXmagicApi != null && avatarData != null) {
            List<AvatarData> avatarDataList = new ArrayList<>();
            avatarDataList.add(avatarData);
            mXmagicApi.updateAvatar(avatarDataList, this);
        }
 }
```

Avatar UIのカスタマイズ

最終更新日：：2023-02-27 14:18:15

Demo UIの説明

実現方式

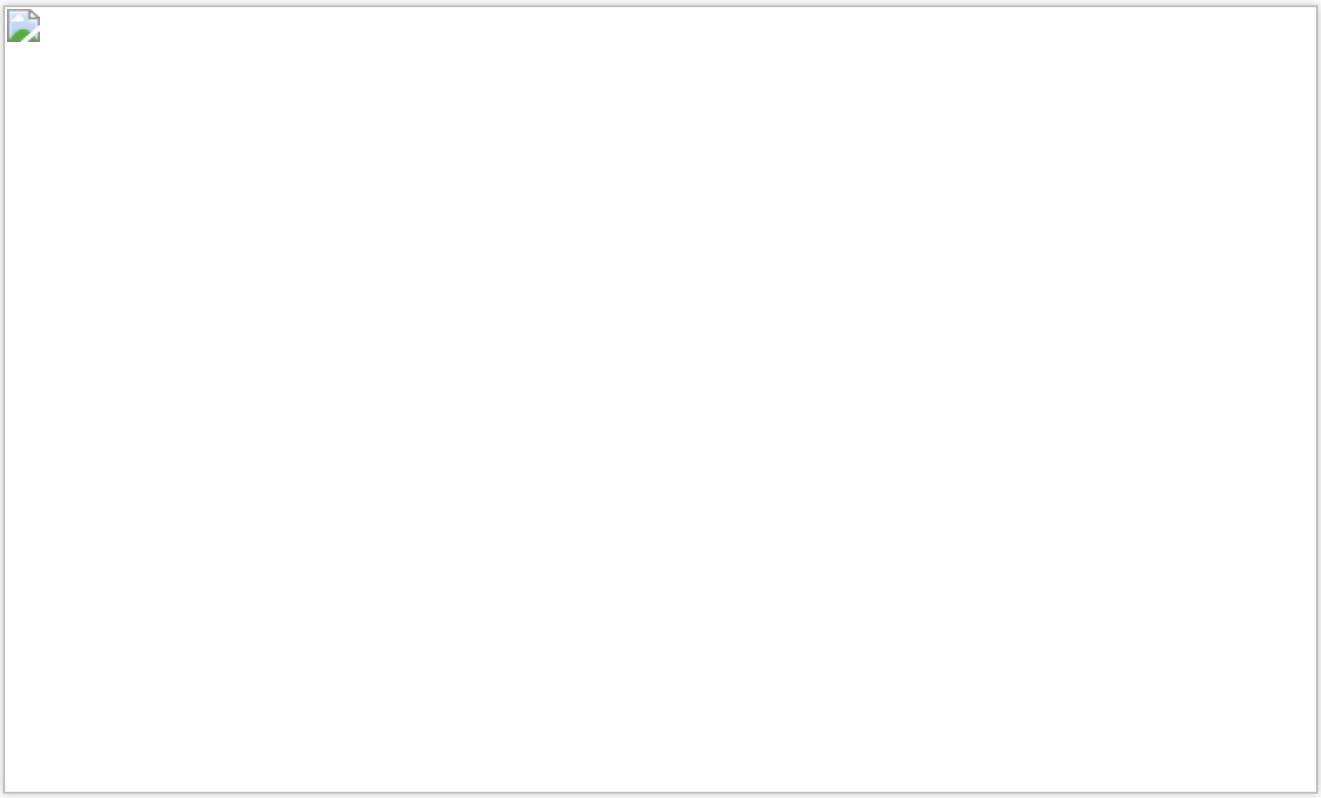
パネル設定情報は任意のパスに配置できます。Demoではassetsに配置し、Demoでパネルファイルを初めて使用する際にインストールディレクトリ下にコピーします。

Json構造とUIパネルの対応関係：

左側のitemが右側ページの第1階層メニューに対応します。headは最初のiconで選択した内容です。

左側の赤枠のsubTabsは右側の第2階層メニューに対応します。

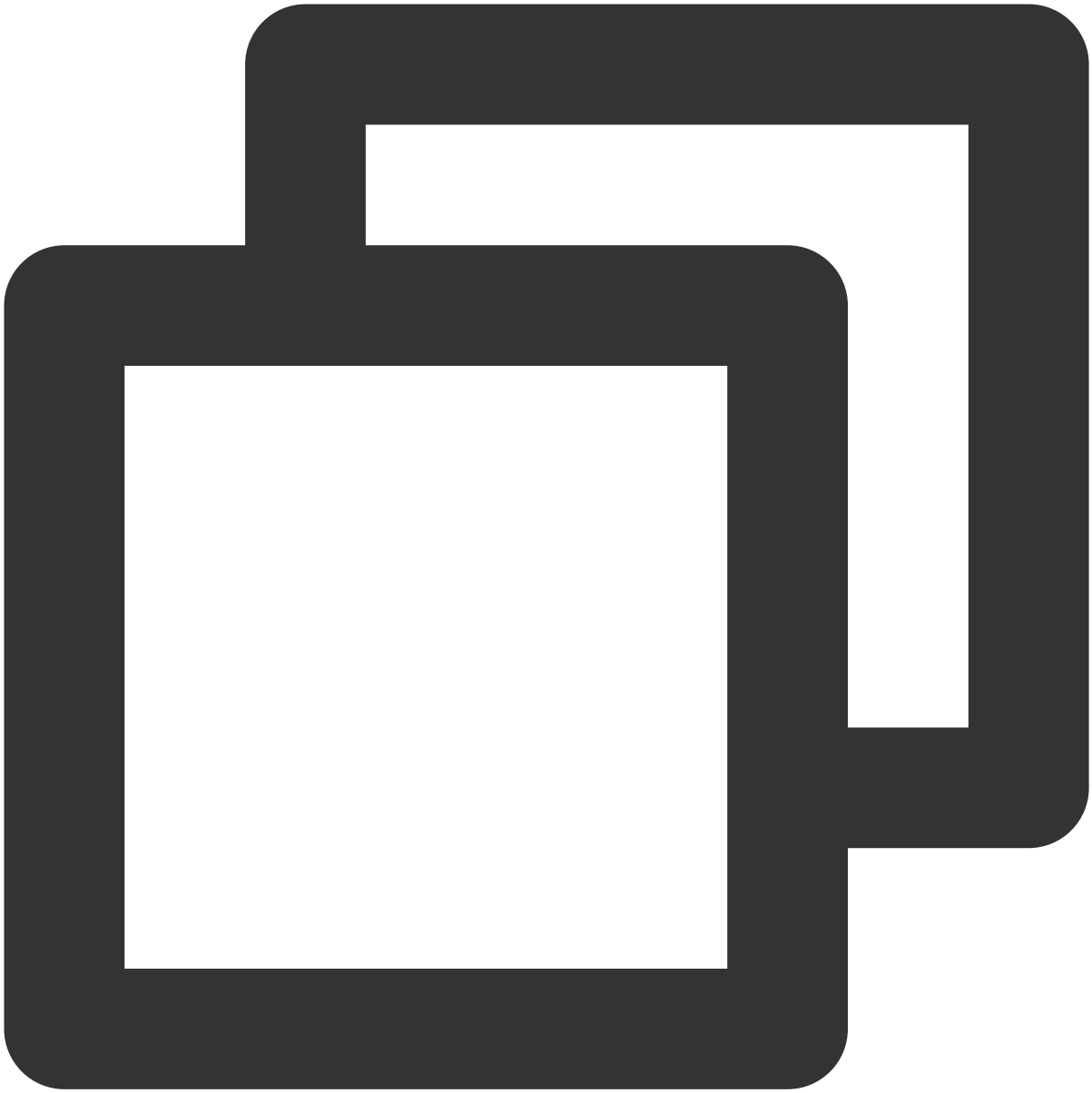
左側のiconのデータはresourcesフォルダ内で設定します。右側に表示されるのはパネルの設定データであり、両者はパネルデータの**categoryでバインドします**。SDKはresourcesフォルダ内のデータを解析し、対応するmapに入力します。mapのkeyはcategoryの値であるため、Demoでは `panel.json` ファイルの解析が完了すると、SDKの提供するメソッドによってデータを取得してバインドできるようになります。



Demoの重要クラスの説明

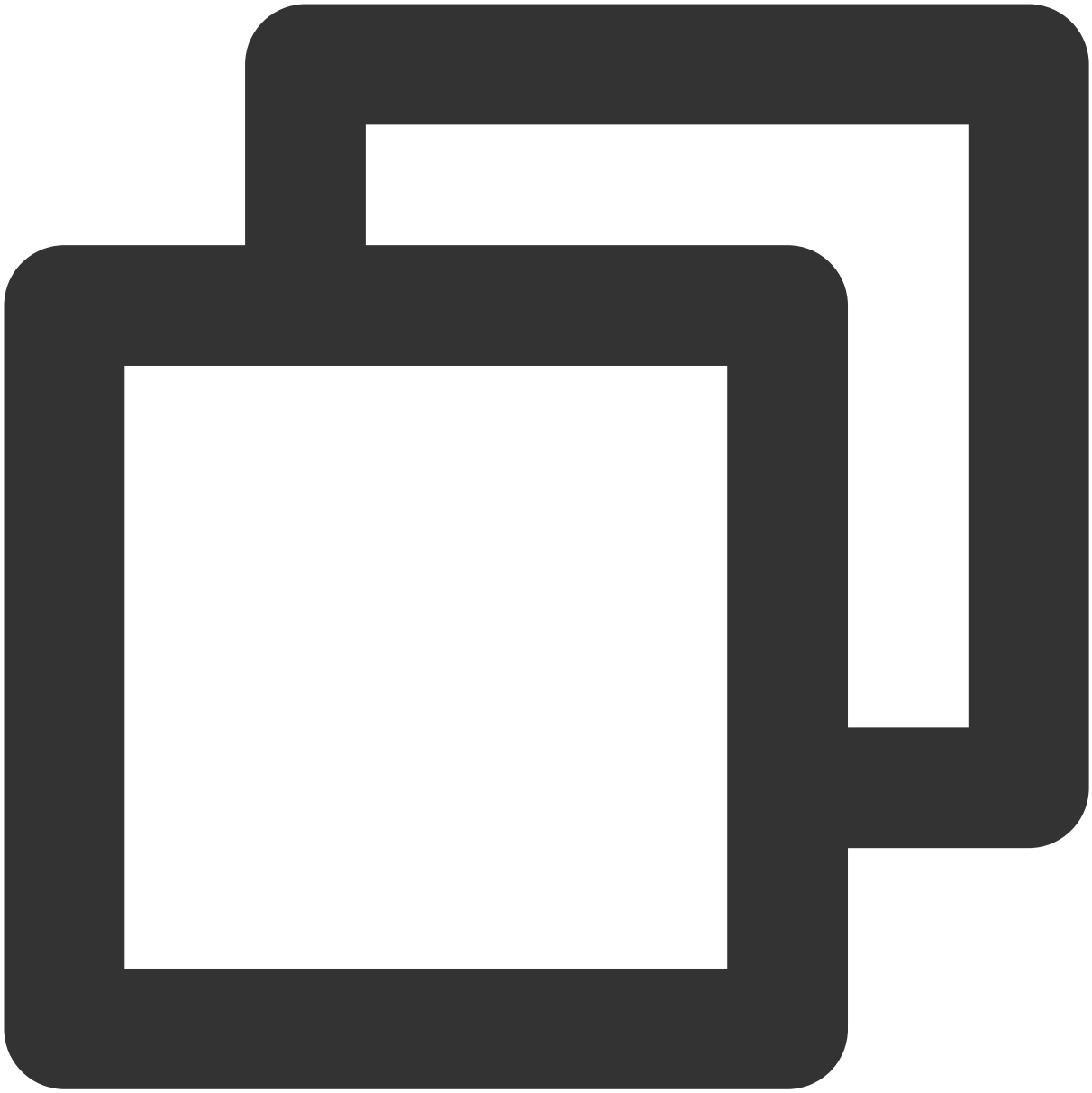
パス： `com.tencent.demo.avater.AvatarResManager.java`

1. Avatarリソースのロード



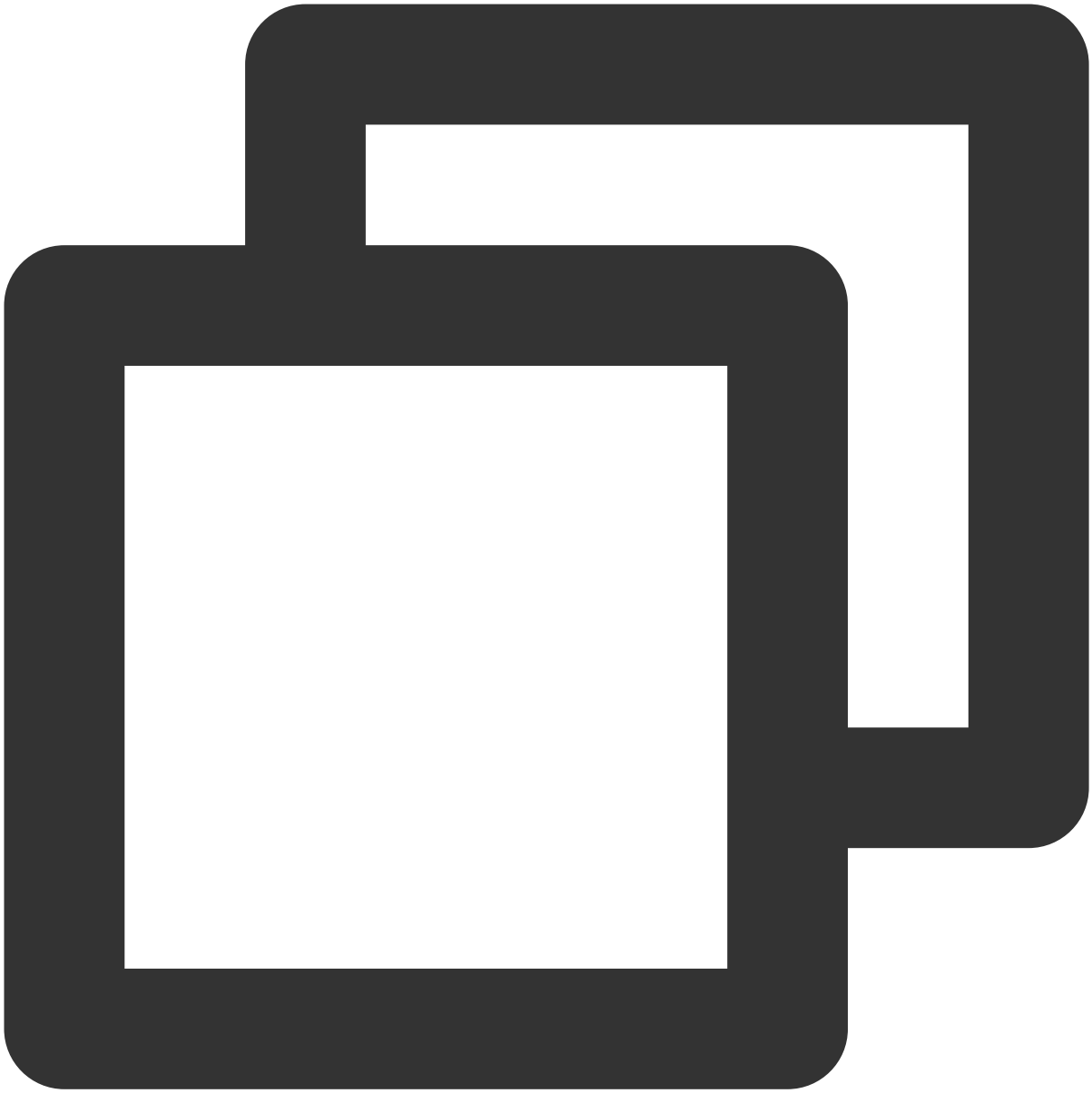
```
/**
     * Avatarリソースのロードに使用します
     *
     * @param xmagicApi      XmagicApiオブジェクト
     * @param avatarResName  名前
     * @param avatarSaveData モデルのデフォルト設定をロードします。ない場合はnullを渡します
     */
    public void loadAvatarRes(XmagicApi xmagicApi, String avatarResName, String ava
```

2. パネルデータの取得



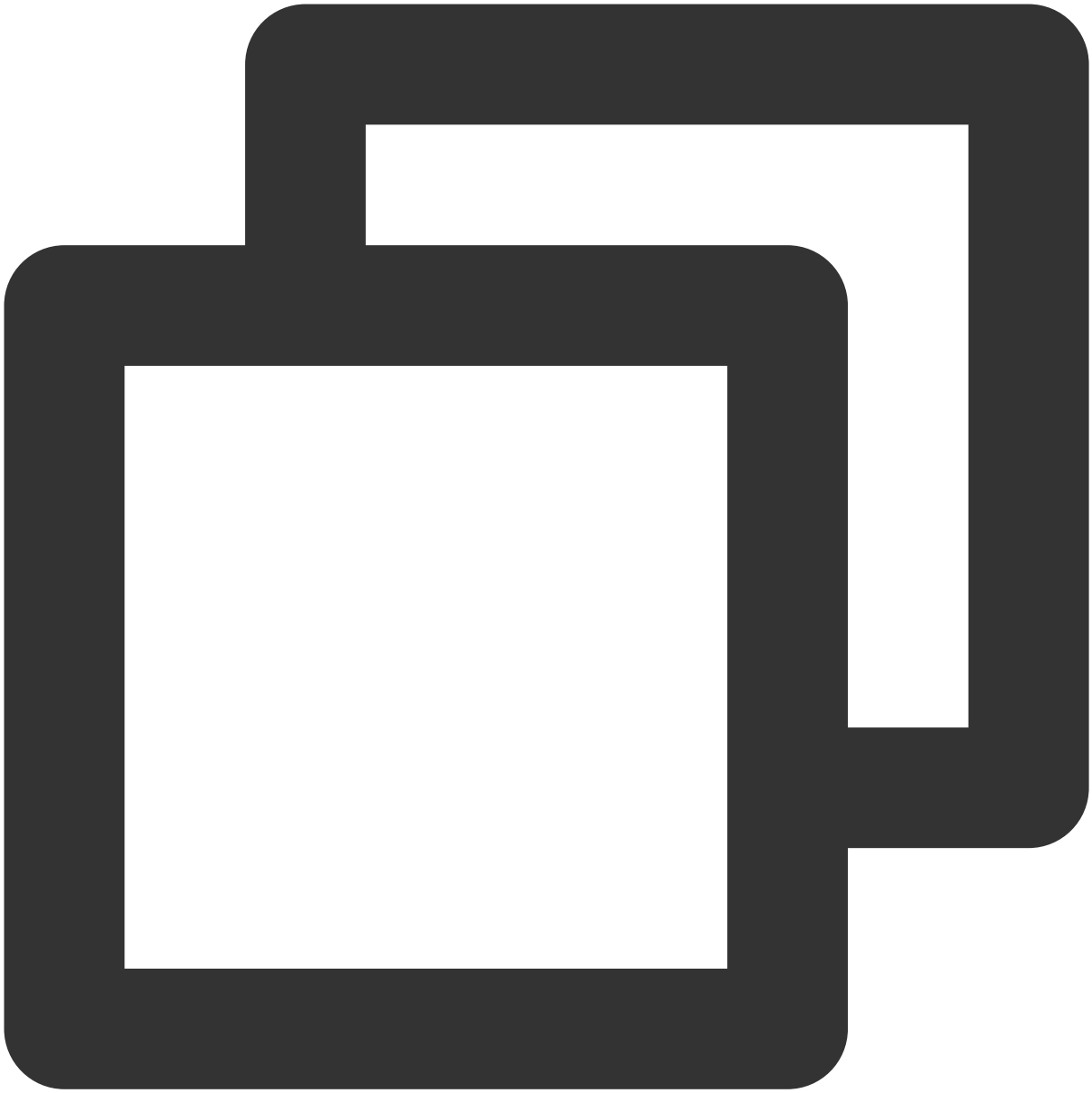
```
/**
     * avatarのパネルデータを取得します。
     *
     * @param avatarResName      avatar素材名
     * @param avatarDataCallBack このメソッドはファイルにアクセスするため、サブスレッド内でフ
     *                           返されるデータにはresourcesフォルダ下のデータが含まれていま
     */
    public void getAvatarData(String avatarResName, String avatarSaveData, LoadAvat
```

3. パネルデータから、ユーザーの設定した属性またはデフォルトの属性を解析します



```
//パネルの設定ファイルから、ユーザーの設定した属性またはデフォルトの属性を解析します
public static List<AvatarData> getUsedAvatarData(List<MainTab> mainTabList)
```

4. モデル切り替え背景データの取得



```
/**
     * 対応するplane Configデータを取得します
     *
     * @param avatarResName リソース名
     * @param avatarType    背景タイプ
     * @return
     */
    public AvatarData getAvatarPlaneTypeConfig(String avatarResName, AvatarBgType a
```

付録

MainTab.java、SubTab.java内のフィールドの説明です。

MainTab

フィールド	タイプ	入力必須かどうか	意味
id	String	はい	メインメニューの一意の識別子。メインメニューを区別するために用いられるため、グローバルで一意である必要があります
label	String	いいえ	第1階層メニュー名（Demoでは現在この名称を表示していません
iconUrl	String	はい	画像アドレス。選択していない場合の画像アドレスです
checkedIconUrl	String	はい	画像アドレス。選択している場合の画像アドレスです
subTabs	SubTabタイプリスト	はい	第2階層メニューリスト

SubTab

フィールド	タイプ	入力必須かどうか	意味
label	String	はい	第2階層メニュー名
category	String	はい	第2階層メニューのカテゴリータイプ。SDKの `com.tencent.xmagic.avatar.AvatarCategory` クラスで定義します
relatedCategory	String	いいえ	依存関係の識別に使用します。例：髪型と髪色に依存関係があり、髪型変更の際もユーザーが前回設定した髪色を使用する必要があるような場合、髪型内にこのフィールドを設定する必要があります。値は髪色内のcategoryフィールドの値です（依存関係は現時点ではtype値がTYPE_SELECTORタイプのデータにしか存在しません）
avatarDataList	リストタイプ	はい	AvatarIconタイプリストデータ

AvatarItem

フィールド	タイプ	入力必須かどうか	意味
id	String	はい	各属性のID。SDKが返すAvatarDataデータ内のIDに対応します
icon	String	はい	画像アドレスまたはARGB色値（”#FF0085CF“）
type	Int	はい	UI表示タイプ。 `AvatarData.TYPE_SLIDER` はスライダータイプ、 `AvatarData.TYPE_SELECTOR` はiconタイプです
selected	boolean	はい	typeがAvatarData.TYPE_SELECTORタイプの場合、このフィールドはそのitemが選択されているかどうかを表します
downloadUrl	String	いいえ	設定ファイルのダウンロードアドレス。設定ファイルの動的ダウンロードに使用します
category	String	はい	SubTab内のcategoryと同義です
labels	Map<String, String>	いいえ	typeがAvatarData.TYPE_SLIDERタイプの場合は値があり、パネルの左側に表示されるlabelに配置されます
avatarData	AvatarData	はい	SDKはアバター制作属性操作クラスを定義しています

Avatar SDKの説明

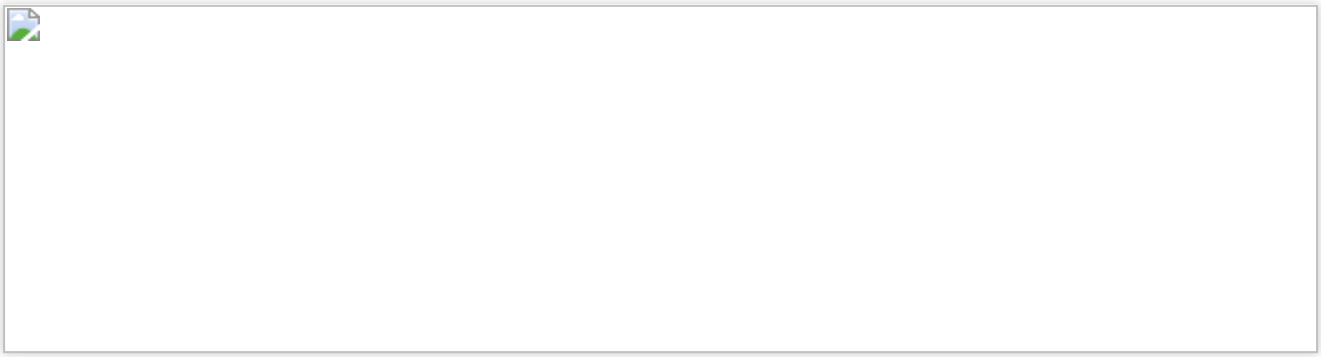
最終更新日：：2023-02-27 14:18:15

SDKへのアクセス

SDKのダウンロード、接続、認証、Demoクイックスタートについては、Tencent Effectの独立した統合をご参照ください。

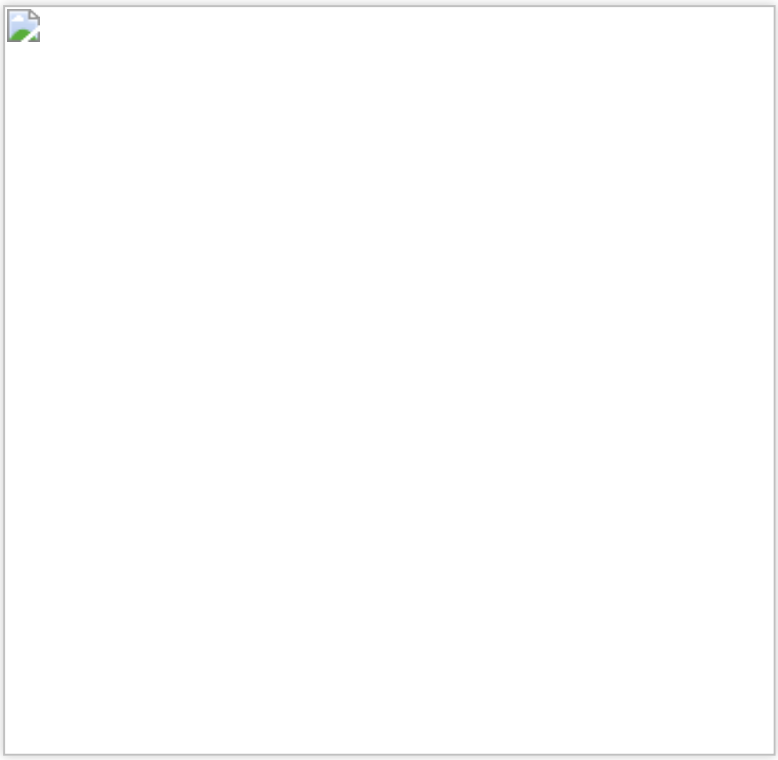
アバター制作素材の準備

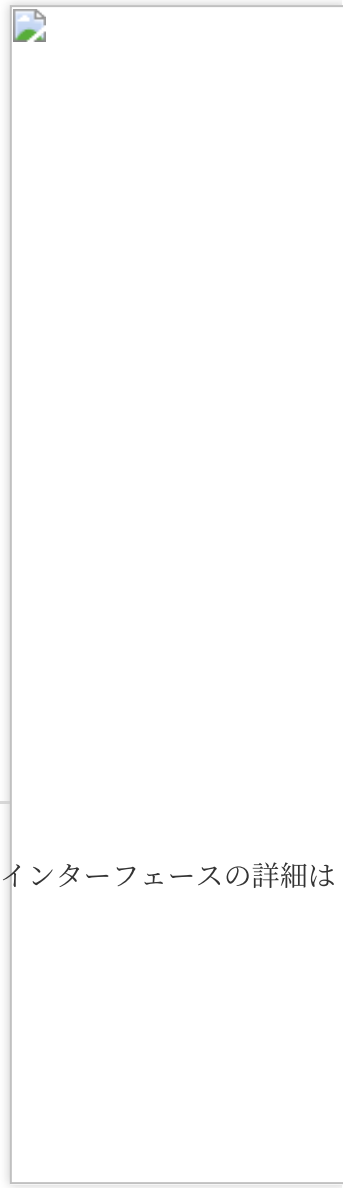
現在Tencentでは、SDKと共にいくつかのアバター制作、着せ替え素材をご提供しています。素材はSDKの解凍後の `MotionRes/avatarRes` ディレクトリ内にあります。他の動的エフェクト素材と同様に、プロジェクトのassetsディレクトリにcopyする必要があります。



アバター制作フローとSDKインターフェース

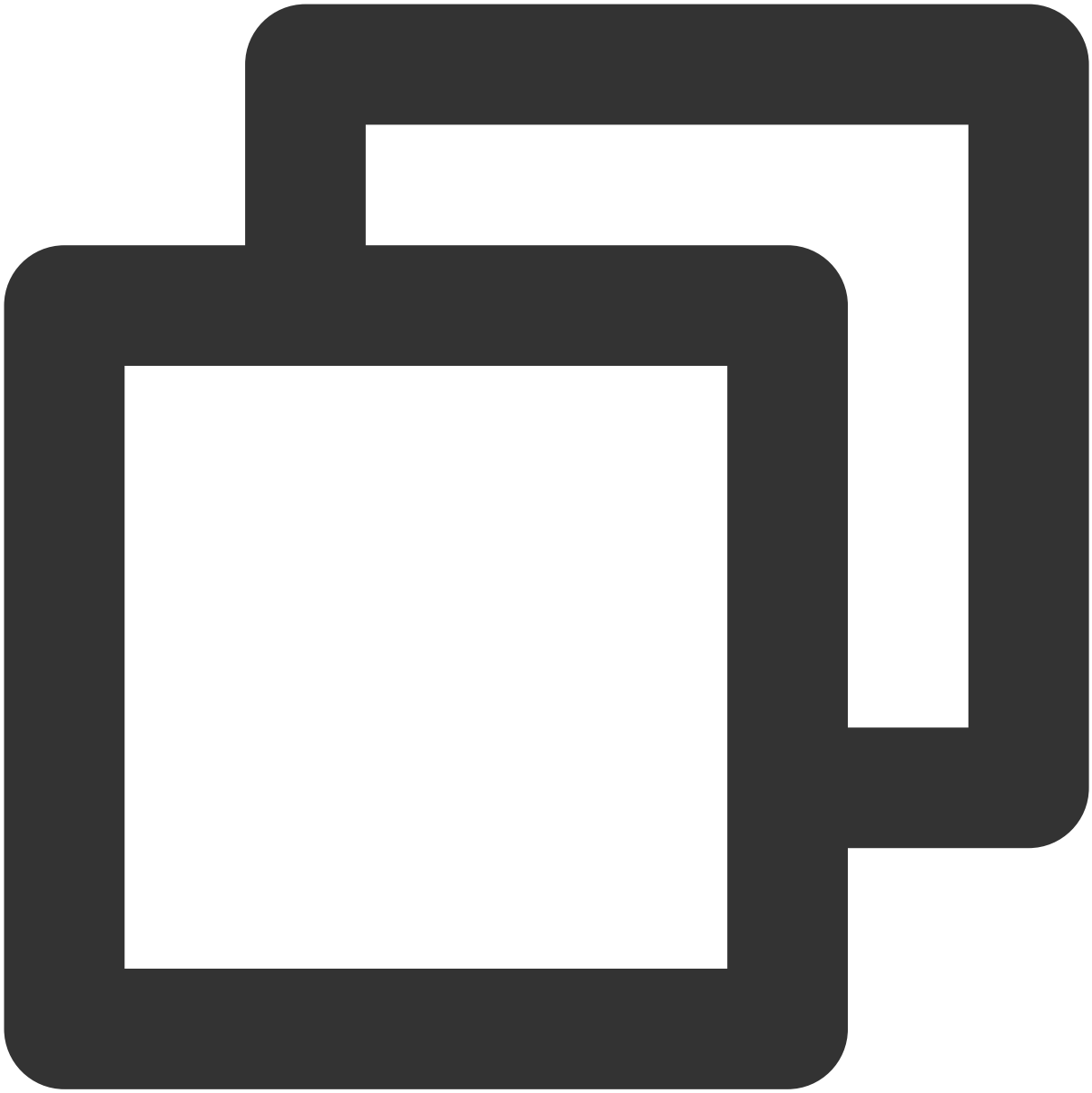
アバター制作フロー	写真撮影アバター制作フロー





XMagicApiのロードデータ、アバター制作、エクスポート設定、写真撮影アバター制作インターフェースの詳細は次のとおりです。

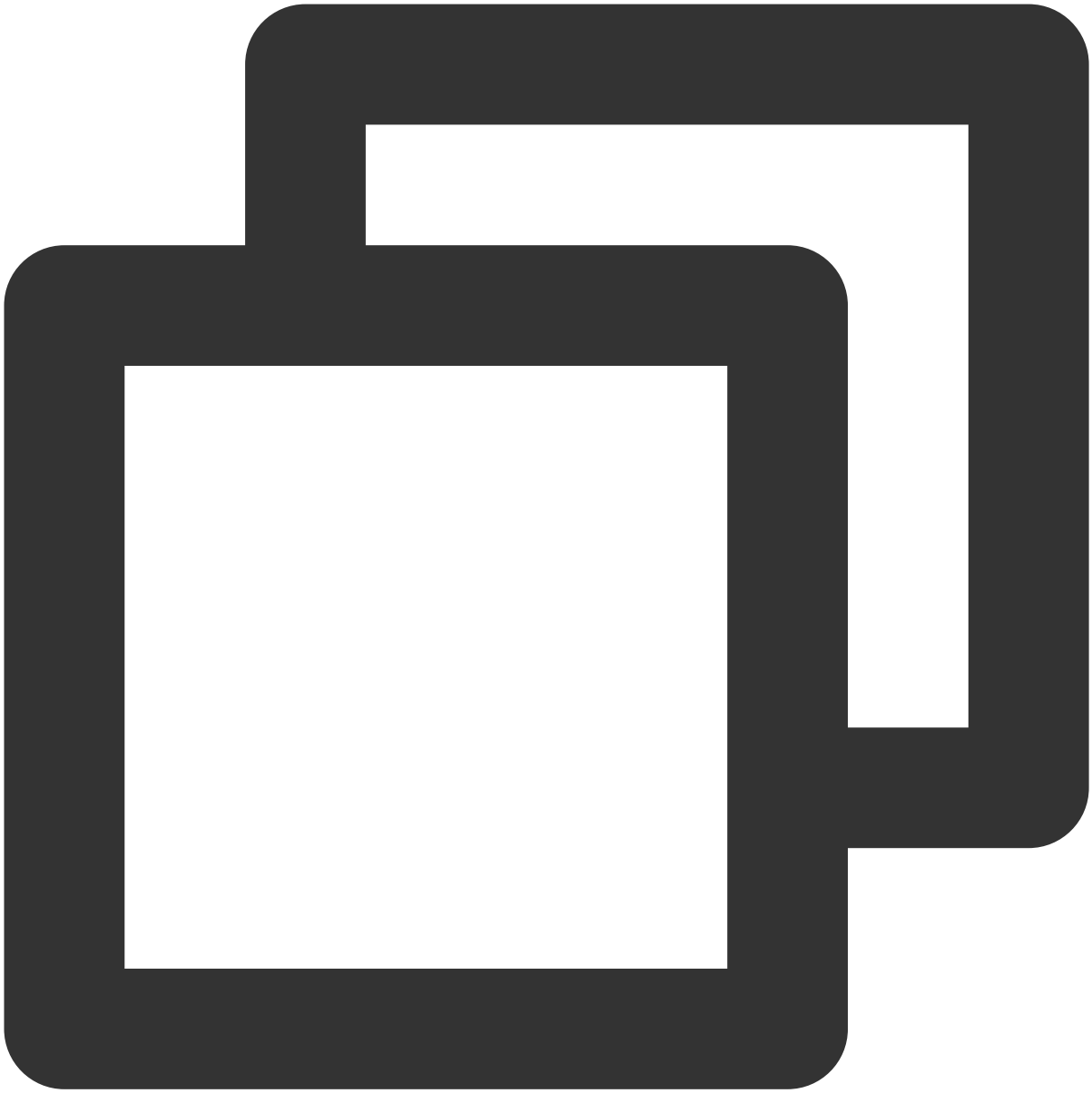
1. Avatar素材のロード（loadAvatar）



```
public void loadAvatar(XmagicProperty<?> property, UpdatePropertyListener updatePro
```

Avatar素材と一般的な動的エフェクト素材のロード方法は類似しています。loadAvatarインターフェースとSDKのupdatePropertyインターフェースは等価です。このためupdatePropertyインターフェースの説明とDemoコードをご参照ください。

2. Avatarソースデータのロード（getAvatarConfig）



```
public static Map<String,List<AvatarData>> getAvatarConfig(String avatarResPath, St
```

入力パラメータ：

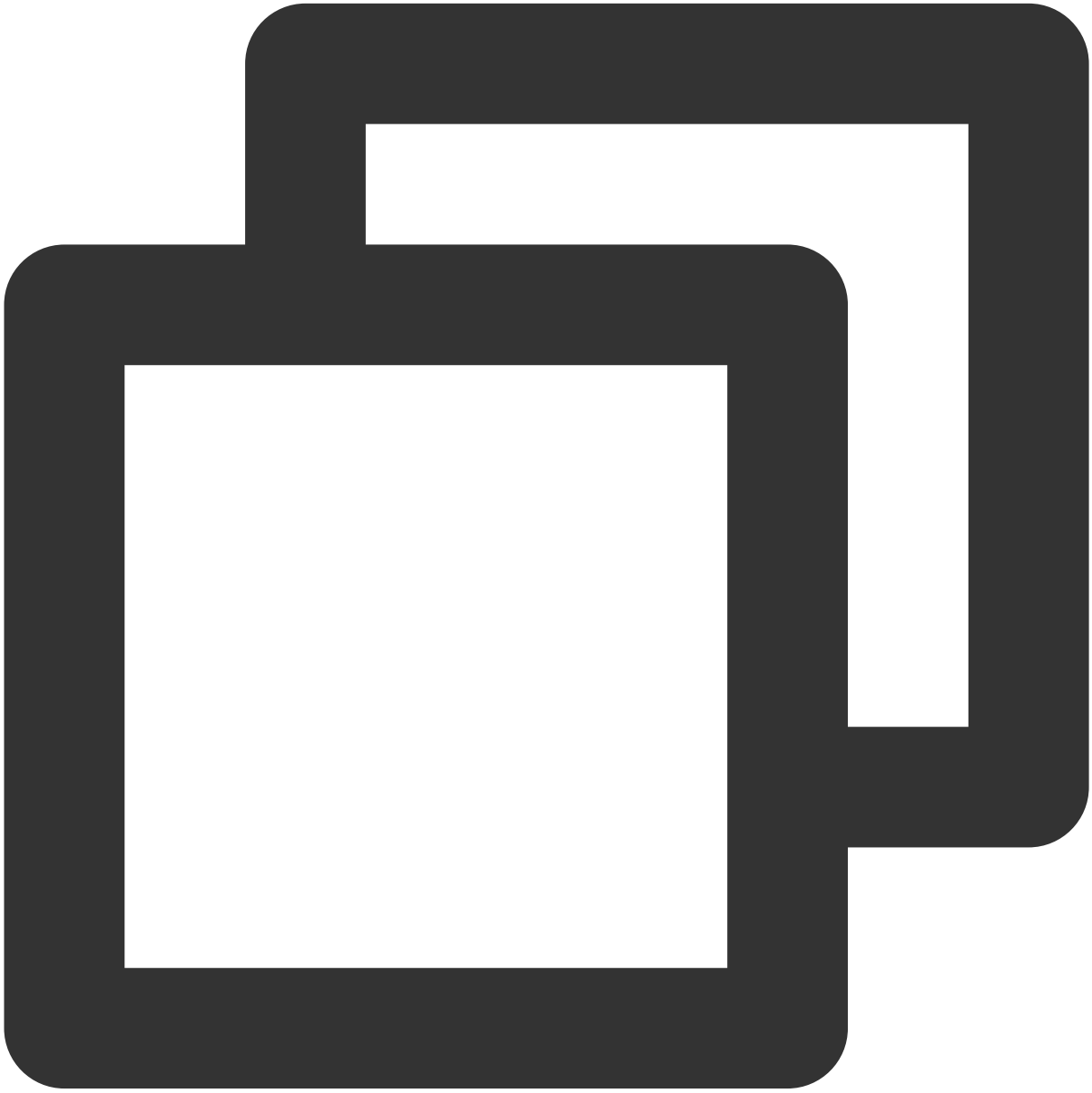
avatarResPath：avatar素材のスマートフォン上の絶対パスです。

例： `/data/data/com.tencent.pitumotiondemo.effects/files/xmagic/MotionRes/avatarRes/animoji_0624` 。

savedAvatarConfigs：ユーザーが前回のアバター制作後に保存したデータであり、JSON形式の文字列です。初めて使用する場合またはユーザーがそれまでに保存したことがない場合、この値はnullとなります。

出力パラメータ：
mapの形式で返します。mapのkeyはデータのcategoryです。詳細についてはAvatarCategoryクラスをご覧ください。mapのvalueはこのcategory下のすべてのデータです。アプリケーション層でこのmapを取得した後、必要に応じて希望のUIスタイルを表示します。

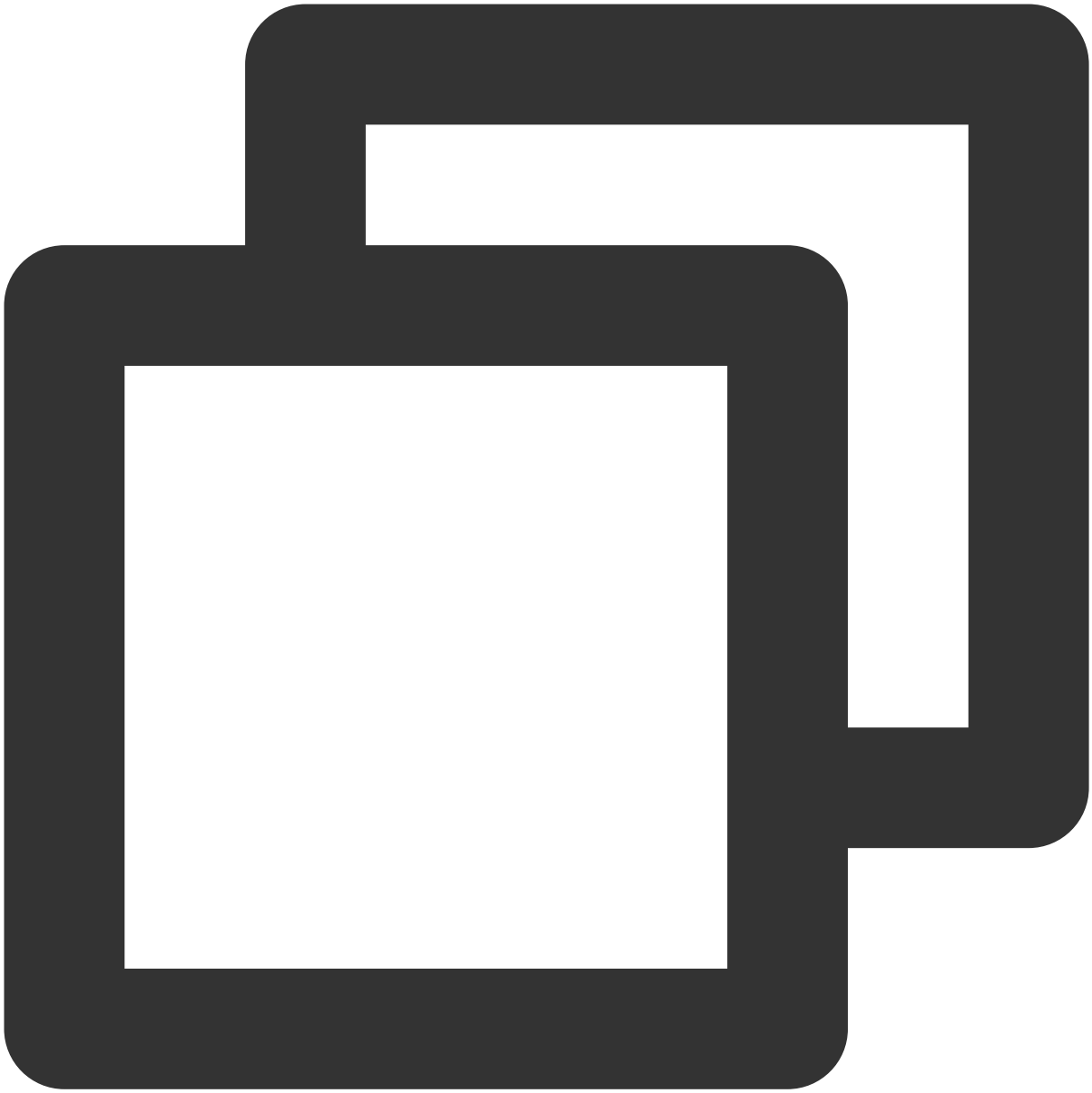
3. アバター制作、着せ替え（updateAvatar）



```
public void updateAvatar(List<AvatarData> avatarDataList, UpdateAvatarConfigListene
```

呼び出すと現在の素材のプレビューキャラクターをリアルタイムに更新します。1つのAvatarDataオブジェクトは1つのアトミック設定であり（髪型変更など）、一度に複数のアトミック設定を渡すことができます（髪型と髪色の両方を変更するなど）。このインターフェースは渡されるAvatarDataの有効性をチェックし、有効なものはSDKに設定し、無効なデータにはcallbackを返します。
例えば髪型の変更を要求しているのに、髪のモデルファイル（AvatarDataのvalueフィールド内に設定）がローカルに見つからない場合、このAvatarDataは無効と判断されます。
また例えば、瞳マップの変更を要求しているのに、マップファイル（AvatarDataのvalueフィールド内に設定）がローカルに見つからない場合、このAvatarDataは無効と判断されます。

4. アバター制作設定のエクスポート（exportAvatar）



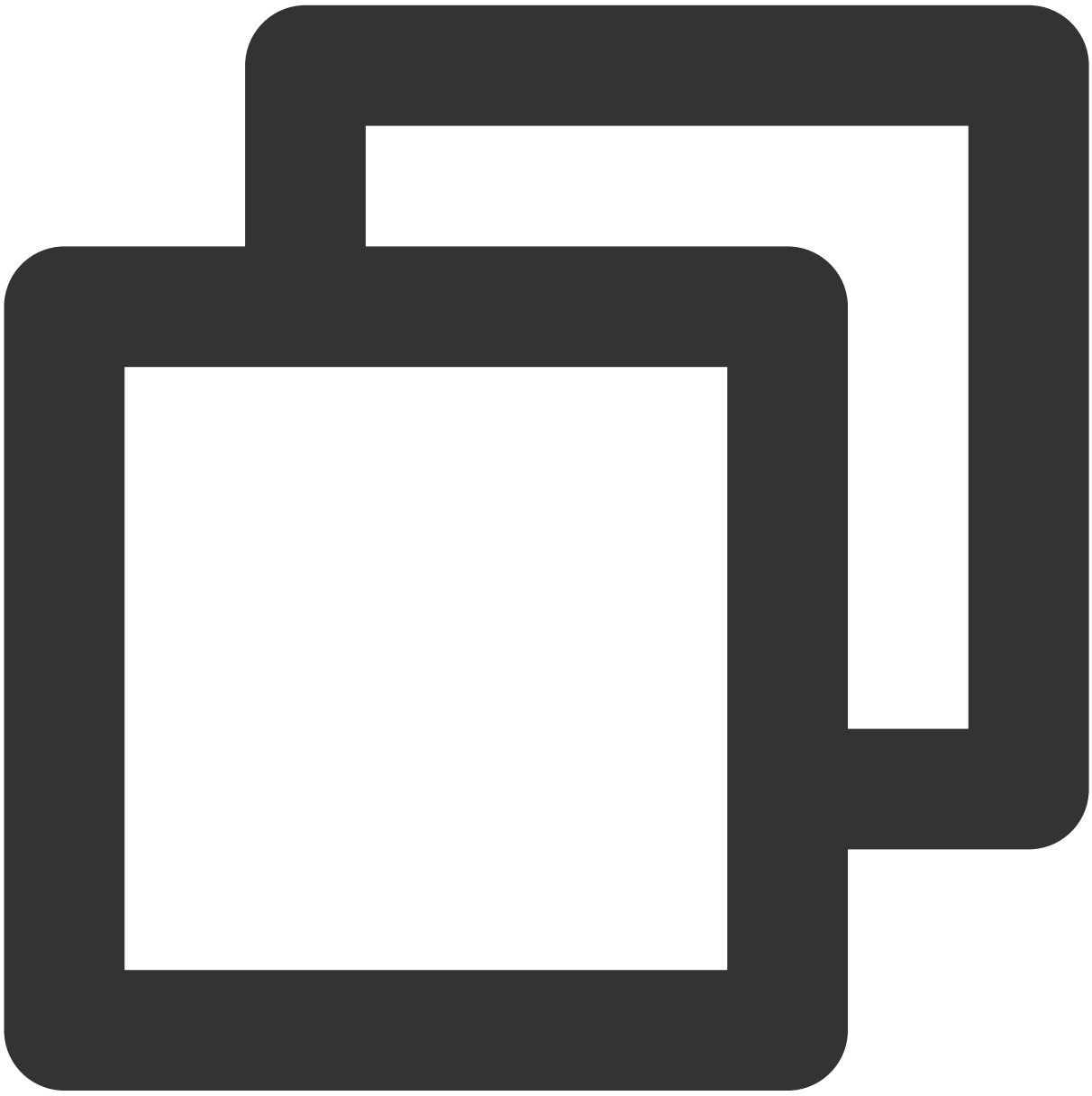
```
public static String exportAvatar(List<AvatarData> avatarDataList)
```

ユーザーがアバターを制作する際、AvatarDataのselectedステータスまたはブレンドシェイプ値を変更する場合があります。制作完了後、新しいフルAvatarDataリストを渡すとjson文字列をエクスポートできます。この文字列はローカルに保存することも、サーバーにアップロードすることもできます。
このエクスポートされた文字列には2つの用途があります。
次回再びXMagicApiのloadAvatarインターフェースでこのAvatar素材をダウンロードする際、このJSON文字列をXMagicPropertyのcustomConfigsフィールドに設定する必要があります。こうすることで、ユーザーが前回作成したキャラクターをプレビューで表示することができます。

上記のように、getAllAvatarDataを呼び出す際にこのパラメータを渡すことで、Avatarソースデータ内の選択状態およびブレンドシェイプ値を変更できるようにする必要があります。

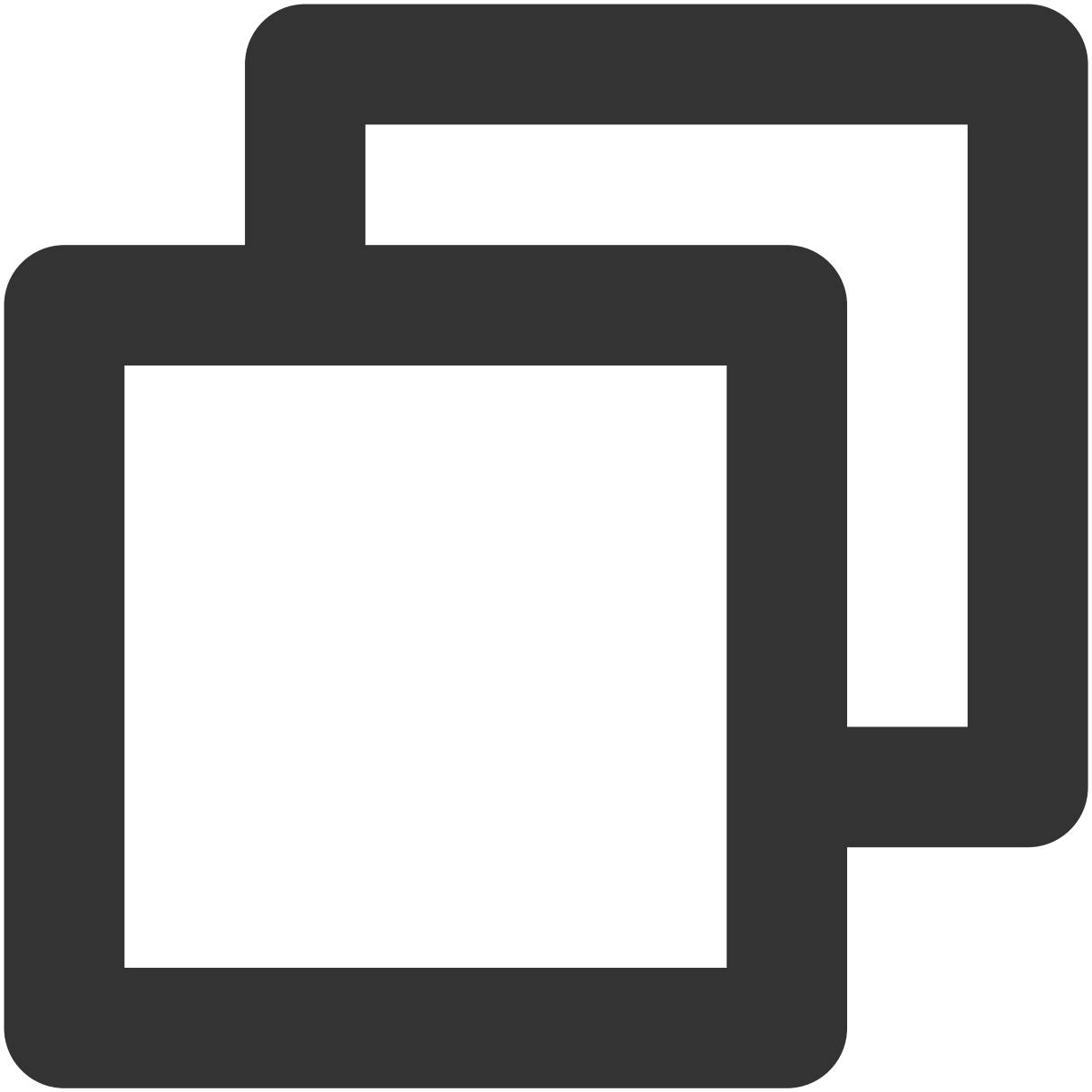
5. 写真撮影、アバター制作（createAvatarByPhoto）

このインターフェースにはネットワーク接続が必要です。



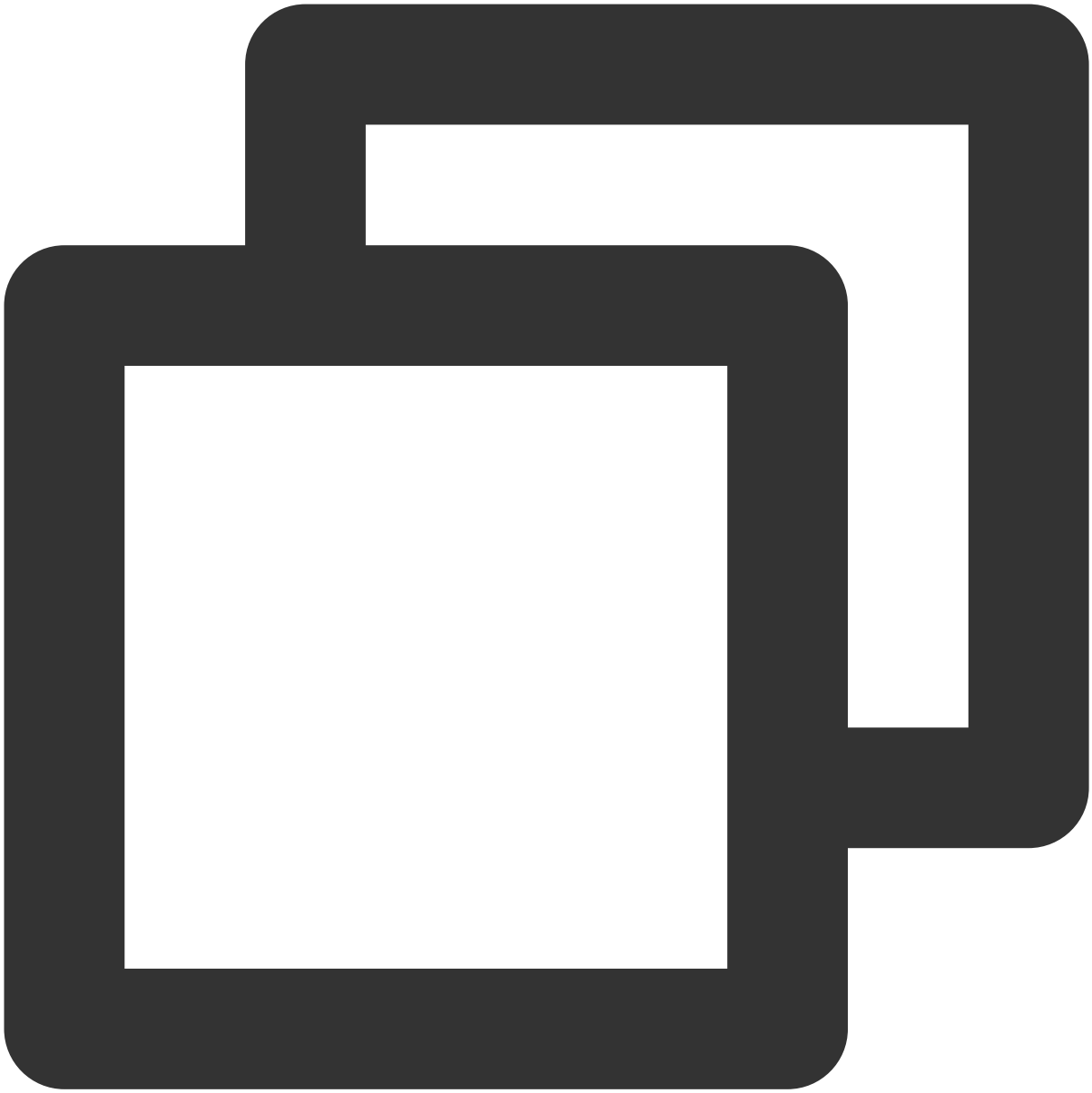
```
public void createAvatarByPhoto(String photoPath, List<String> avatarResPaths, bool
            final FaceAttributeListener faceAttributeListener)
```

photoPath：写真のパスです。顔が確実に画面の中央に位置するようにしてください。画面内に含まれる顔は1つだけにすることをお勧めします。複数の顔がある場合、SDKはその中の1つを選択します。写真の短辺は500px以上を推奨します。これより小さいと認識効果に影響する可能性があります。
avatarResPaths：複数のAvatar素材セットを渡すことができます。SDKは写真の分析結果に基づいて、最も適した素材のセットを選択し、自動的にアバター制作を行います。注：現在は1セットのみをサポートしています。複数のセットを渡した場合、SDKは最初の1セットのみを使用します。
isMale：男性かどうか。この属性は現在使用されていませんが、今後SDKが最適化される場合を考え、正確に渡すことをお勧めします。
faceAttributeListener：失敗した場合は、`void onError(int errCode, String msg)` をコールバックします。成功した場合は、`void onFinish(String matchedResPath, String srcData)` をコールバックします。 最初のパラメータはマッチしたAvatar素材パス、2番目のパラメータはマッチ結果です。上記で述べたexportAvatarインターフェースからの戻り値は同じ意味となります。
onErrorのエラーコードは `FaceAttributeHelper.java` で定義します。具体的には次のとおりです。



```
public static final int ERROR_NO_AUTH = 1;//権限がありません
public static final int ERROR_RES_INVALID = 5;//渡されたAvatar素材のパスが無効です
public static final int ERROR_PHOTO_INVALID = 10;//写真の読み取りに失敗しました
public static final int ERROR_NETWORK_REQUEST_FAILED = 20;//ネットワークリクエストに失
public static final int ERROR_DATA_PARSE_FAILED = 30;//ネットワークから返されたデータの
public static final int ERROR_ANALYZE_FAILED = 40;//顔の分析に失敗しました
public static final int ERROR_AVATAR_SOURCE_DATA_EMPTY = 50;//Avatarソースデータのロ
```

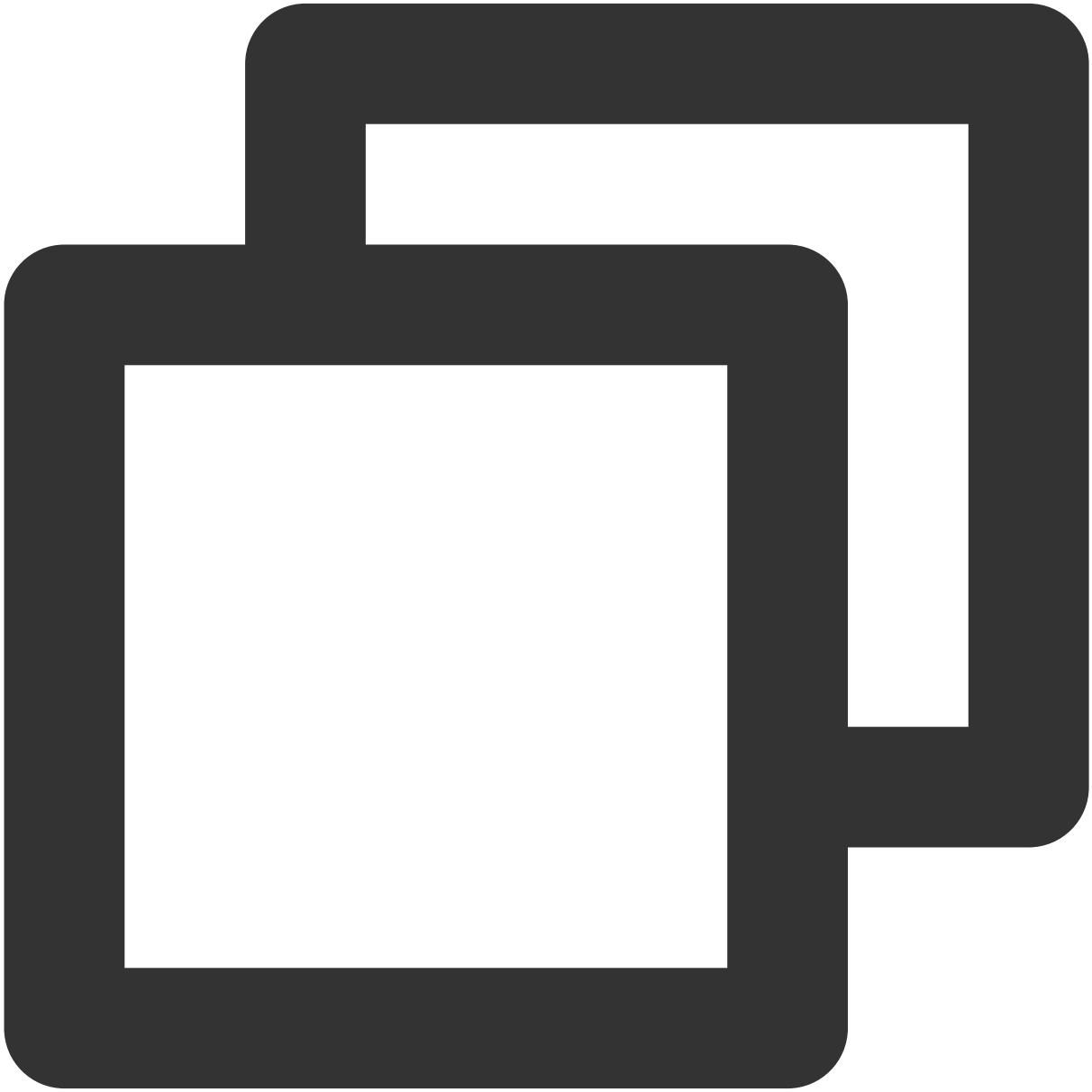
6. ダウンロードした設定ファイルを対応するフォルダ内に配置する（addAvatarResource）



```
public static Pair<Integer, List<AvatarData>> addAvatarResource(String resourceRoot
```

このインターフェースは主にAvatarパーツの動的ダウンロードのシナリオに用いられます。例としては、Avatar素材内に10種類の髪型があり、その後1種類の髪型をクライアントに動的に送信したい場合、ダウンロード完了後に圧縮パッケージを取得してからこのインターフェースを呼び出し、圧縮パッケージのパスをSDKに渡します。SDKはこの圧縮パッケージを解析し、対応するcategoryディレクトリ下に保存します。getAvatarConfigインターフェースを次に呼び出す際、SDKは新たに追加されたこのデータを解析することができます。
パラメータの説明：

resourceRootPath：Avatar素材のルートディレクトリです。例えば/data/data/com.tencent.pitumotiondemo.effects/files/xmagic/MotionRes/avatarRes/animoji_0624などです
category：ダウンロードしたこのパーツのカテゴリーです
zipFilePath：ダウンロードしたzipパッケージのアドレスです
インターフェースは `Pair<Integer, List<AvatarData>>` を返します。pair.firstはエラーコード、pair.secondは新たに追加されたデータの集合です。
エラーコードは次のとおりです。

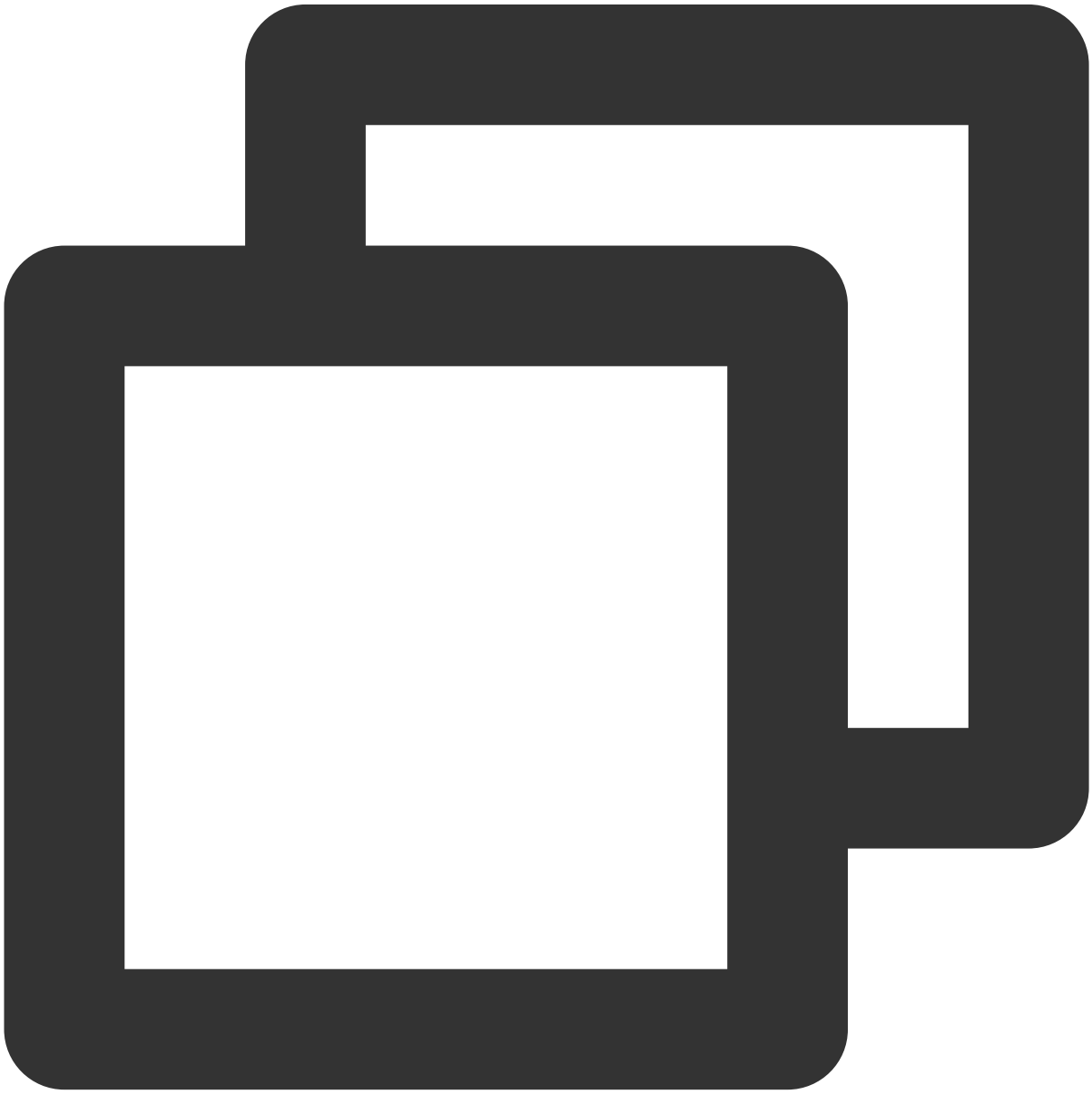


```
public interface AvatarActionErrorCode {
    int OK = 0;
```

```
        int ADD_AVATAR_RES_INVALID_PARAMS = 1000;
        int ADD_AVATAR_RES_ROOT_RESOURCE_NOT_EXIST = 1001;
        int ADD_AVATAR_RES_ZIP_FILE_NOT_EXIST = 1002;
        int ADD_AVATAR_RES_UNZIP_FILE_FAILED = 1003;
        int ADD_AVATAR_RES_COPY_FILE_FAILED = 1004;
    }
```

7. AvatarDataの呼び出し

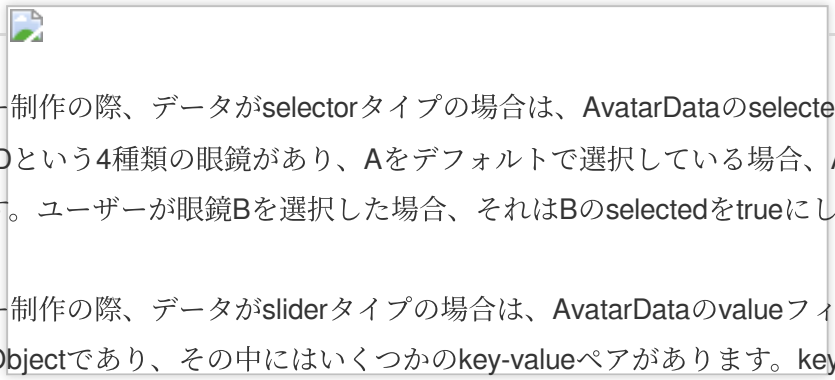
上記のインターフェースのコアはすべてAvatarDataクラスです。その主な内容は次のとおりです。

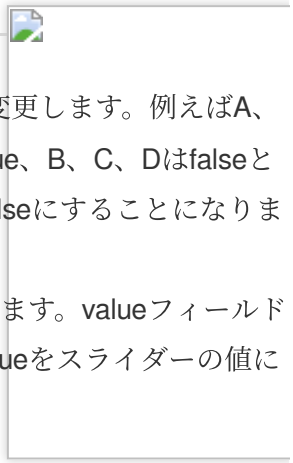


```
public class AvatarData {
    /**
     * selectorタイプのデータです。例えば眼鏡の場合、さまざまな種類の眼鏡があっても、使用する際に
     */
    public static final int TYPE_SELECTOR = 0;

    /**
     * スライダー調整タイプのデータです。例えば頬の幅を調整するなどです。
     */
    public static final int TYPE_SLIDER = 1;

     // 例えば、輪郭、目の微調整などです。AvatarCategory.javaでは標準のcategoryを定義し、それ
    //空にすることはできません。
    public String category;

    // 具体的な各itemまたは各微調整項目を識別します。
    // 例えば各眼鏡にはすべて個々のidがあり、微調整項目の各グループにもすべて個々のidがあります。
    //空にすることはできません。
    public String id;

    //TYPE_SELECTORまたはTYPE_SLIDER
    public int type;

    //selectorタイプの場合は、現在選択されているものの有無を表します
    public boolean selected = false;

    //各アイコンまたは微調整項目の各グループのバックグラウンドはすべて具体的な設定の詳細である、次
    public String entityName;
    public String action;
    public JsonObject value;
}
```

1つのAvatarDataオブジェクトは1つのアトミック設定です（髪型変更、 頬の調整など）。

髪型変更	頬の調整





アバター制作の際、データがselectorタイプの場合は、AvatarDataのselectedフィールドを変更します。例えばA、B、C、Dという4種類の眼鏡があり、Aをデフォルトで選択している場合、Aのselectedはtrue、B、C、Dはfalseとなります。ユーザーが眼鏡Bを選択した場合、それはBのselectedをtrueにし、A、C、Dをfalseにすることになります。
アバター制作の際、データがsliderタイプの場合は、AvatarDataのvalueフィールドを変更します。valueフィールドはJsonObjectであり、その中にはいくつかのkey-valueペアがあります。key-valueの中のvalueをスライダーの値に変更するだけです。

AvatarDataの高度な説明

AvatarDataの中で、アバター制作において重要な役割を果たすものはentityName、action、valueの3つのフィールドです。これら3つのフィールドの値は、SDKが素材設定を解析する際に自動的に入力されます。多くの場合、この3つのフィールドの意味を理解する必要はありませんが、UI層で表示する際、スライダータイプの場合は、value内のブレンドシェイプkey-valueを解析し、UIの操作と対応させる必要があります。
このうち、AvatarDataの要素はentityName、actionおよびvalueのフィールドに分けられます

entityNameフィールド

アバター制作の際は、どの部位（顔、目、髪、トップス、靴など）を制作するかを明確に指定する必要があります。entityNameフィールドはこれらの体の部位の名前を記述するためのものです。

actionおよびvalueフィールド

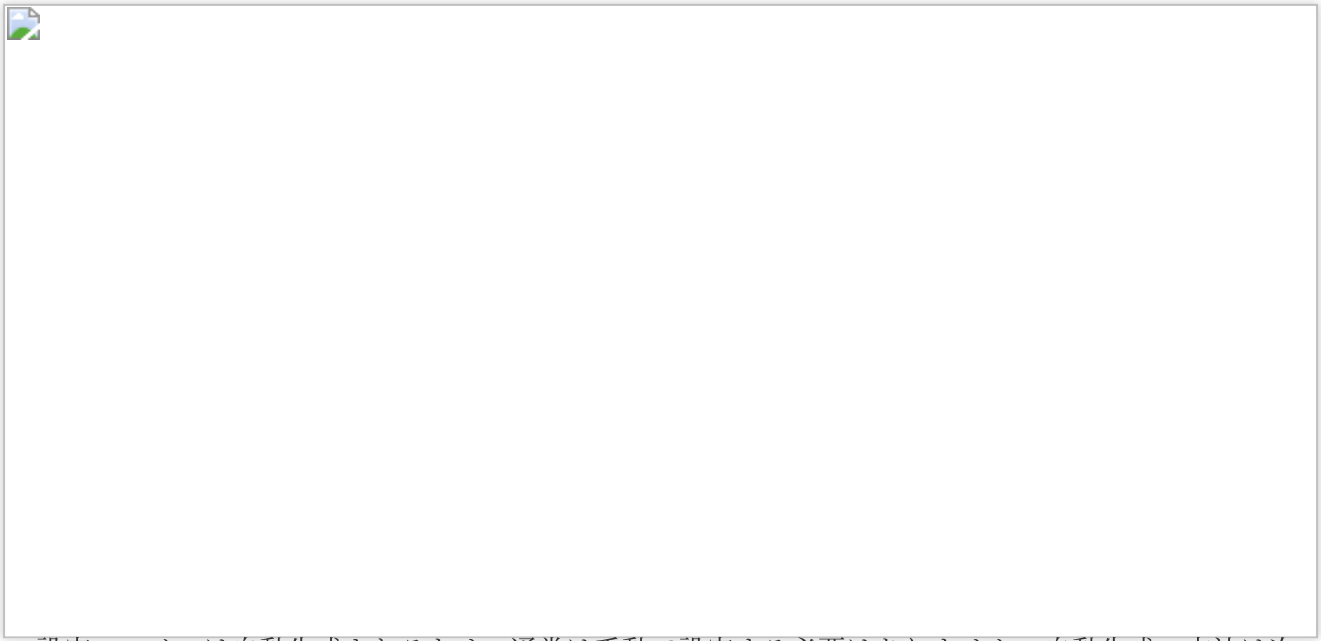
actionフィールドはentityNameに対してどの操作（action）を実行するかを表します。SDK内では5種類のactionを定義し、いずれも `AvatarAction.java` で定義します。各actionの意味およびvalueの要件は次のとおりです。

action	意味	valueの要件
changeColor	現在のマテリアルの色を変更します。基本色、放射光色などの色属性が含まれます	JsonObjectタイプ。入力必須です。素材制作ツールによって自動生成されます。
changeTexture	現在のマテリアルのマップを変更します。色のテクスチャマップ、金属の粗さのテクスチャマップ、AOテクスチャマップ、法線テクスチャマップ、放射光テクスチャマップなどが含まれます	JsonObjectタイプ。入力必須です。素材制作ツールによって自動生成されます。
shapeValue	blendShape値を変更します。通常、	JsonObjectタイプ。このうちkeyはブレンド

	顔の細部のブレンドシェイプの微調整に用いられます	シェイプ名、valueはfloatタイプの値です。入力必須です。素材制作ツールによって自動生成されます。
replace	サブモデルの置き換え（眼鏡、髪型、衣服などの置き換えなど）	JsonObjectタイプ。内部には新しいサブモデルの3D変換情報、モデルパス、マテリアルパスを記述しています。現在の位置にあるサブモデルを非表示にしたい場合はnullを使用します。素材制作ツールによって自動生成されます。
basicTransform	位置、回転、ズームを調整します。通常はカメラの遠近、角度の調整に用いることで、モデルの全身と半身アングルの切り替えを実現します	JsonObjectタイプ。入力必須です。素材制作ツールによって自動生成されます。

Avatar制作・着せ替えデータの設定

avatar属性設定はresourcesフォルダ下に配置します（パス：`素材/custom_configs/resources`）。



これらの設定ファイルは自動生成されるため、通常は手動で設定する必要はありません。自動生成の方法は次のとおりです。

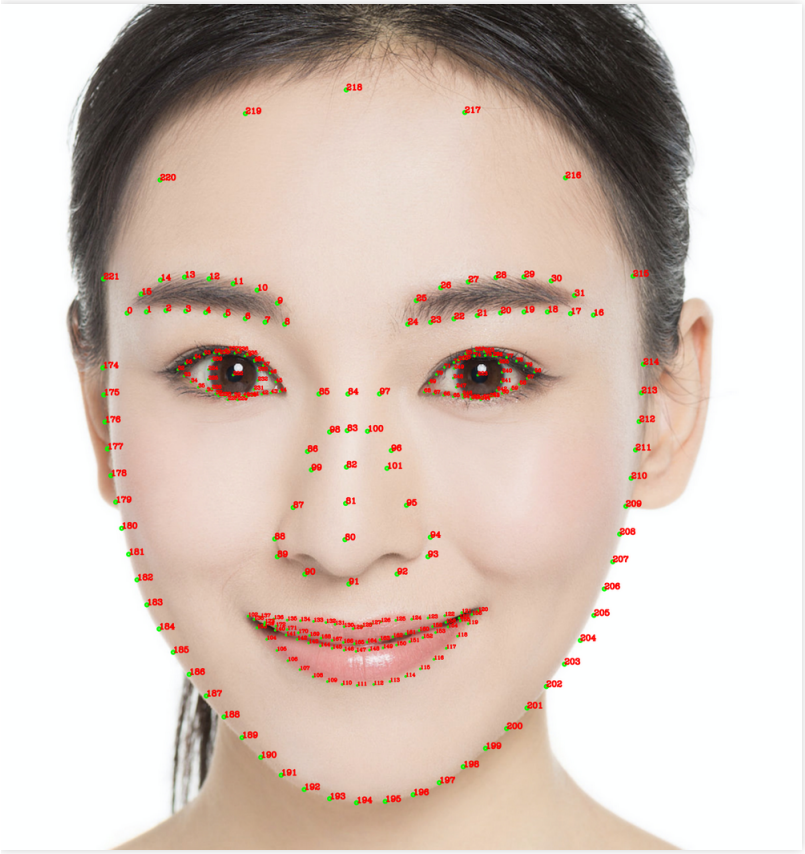
設計者は設計仕様に従い、TencentEffectStudioを使用してキャラクターのセットを設計後、Tencentの提供するresource_generator_guiというApp（現在はMacOSプラットフォームのみサポートしています）を実行するだけでこの設定を自動生成できます。詳細については設計仕様の説明をご参照ください。

アトミック機能統合ガイド
顔ポイント位置統合ガイド

最終更新日：：2022-12-15 11:30:53

顔検出（顔の枠外へのはみ出し、複数の顔、顔部分の遮蔽の認識）では、顔の256個の重要特徴点の位置を認識し出力します。

顔の256ポイント対応インデックス図



iOSインターフェースの説明

iOS統合ガイド

iOSのSDKの統合ガイドについては、Tencent Effectの独立した統合をご参照ください。

Xmagicインターフェースのコールバック登録

```
/// @brief SDKイベント監視インターフェース
/// @param listener イベントリスナーコールバックです。主に、AIイベント、Tips表示イベント、Assetイベントがあります
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;
```

YTSDKEventListenerコールバック説明

```
#pragma mark - イベントコールバックインターフェース
/// @brief SDK内部イベントコールバックインターフェース
@protocol YTSDKEventListener <NSObject>
/// @brief YTDataUpdateイベントコールバック
/// @param event NSString*フォーマットのコールバック
- (void)onYTDataEvent:(id _Nonnull)event;
/// @brief AIイベントコールバック
/// @param event dictフォーマットのコールバック
- (void)onAIEvent:(id _Nonnull)event;
/// @brief 表示イベントコールバック
/// @param event dictフォーマットのコールバック
- (void)onTipsEvent:(id _Nonnull)event;
/// @brief リソースパックイベントコールバック
/// @param event stringフォーマットのコールバック
- (void)onAssetEvent:(id _Nonnull)event;
@end
```

コールバックの設定に成功すると、各フレームの顔イベントごとに次のコールバックを行います。

```
- (void)onYTDataEvent:(id _Nonnull)event;
```

コールバックdataはJSON形式のデータで、具体的な意味は次のとおりです（256のポイントは上図の位置に対応します）。

```
/// @note フィールド意味リスト
/**
| フィールド | タイプ | 値の範囲 | 説明 |
| :---- | :---- |:---- | :---- |
| trace_id | int | [1,INF) | 顔id。連続ストリーム取得の過程で、idが同一であれば同じ顔であると認識できます |
| face_256_point | float | [0,screenWidthまたはscreenHeight] | 計512個。顔の256個の重要特徴点であり、画面左上隅が(0,0)です |
| face_256_visible | float | [0,1] | 顔の256重要特徴点の可視度 |
| out_of_screen | bool | true/false | 顔が枠外に出ていないか |
| left_eye_high_vis_ratio | float | [0,1] | 左目の特徴点のうち高視認度のものが占める割合 |
| right_eye_high_vis_ratio | float | [0,1] | 右目の特徴点のうち高視認度のものが占める割
```

```
合 |
| left_eyebrow_high_vis_ratio | float | [0,1] | 左眉の特徴点のうち高視認度のものが占める割合 |
| right_eyebrow_high_vis_ratio | float | [0,1] | 右眉の特徴点のうち高視認度のものが占める割合 |
| mouth_high_vis_ratio | float | [0,1] | 口の特徴点のうち高視認度のものが占める割合 |
**/
- (void)onYTDataEvent:(id _Nonnull)event;
```

Androidインターフェースの説明

Android統合ガイド

AndroidのSDKの統合ガイドについての詳細は、Tencent Effectの独立した統合をご参照ください。

Xmagicインターフェースのコールバック登録

顔の特徴点位置情報などのデータのコールバックを設定します。

```
void setYTDataListener(XmagicApi.XmagicYTDataListener ytDataListener)
顔の情報などのデータのコールバックを設定します
public interface XmagicYTDataListener {
void onYTDataUpdate(String data)
}
```

onYTDataUpdateはJSON string構造を返します。最大で5つの顔の情報を返します。

```
{
"face_info":[{
"trace_id":5,
"face_256_point":[
180.0,
112.2,
...
],
"face_256_visible":[
0.85,
...
],
"out_of_screen":true,
"left_eye_high_vis_ratio:1.0,
"right_eye_high_vis_ratio":1.0,
"left_eyebrow_high_vis_ratio":1.0,
```

```
"right_eyebrow_high_vis_ratio":1.0,
"mouth_high_vis_ratio":1.0
},
...
]
}
```

フィールドの意味

フィールド	タイプ	値の範囲	説明
trace_id	int	[1,INF)	顔ID。連続してストリームを取得するとき、IDが同じである場合、同じ顔として認識します
face_256_point	float	[0,screenWidth] または [0,screenHeight]	計512個。顔の256個の重要特徴点であり、画面左上隅が(0,0)です。
face_256_visible	float	[0,1]	顔の256個の重要特徴点の視認度。
out_of_screen	bool	true/false	顔が枠外に出ていないか。
left_eye_high_vis_ratio	float	[0,1]	左目の特徴点のうち高視認度のものが占める割合。
right_eye_high_vis_ratio	float	[0,1]	右目の特徴点のうち高視認度のものが占める割合。
left_eyebrow_high_vis_ratio	float	[0,1]	左眉の特徴点のうち高視認度のものが占める割合。
right_eyebrow_high_vis_ratio	float	[0,1]	右眉の特徴点のうち高視認度のものが占める割合。
mouth_high_vis_ratio	float	[0,1]	口の特徴点のうち高視認度のものが占める割合。

パラメータ

パラメータ	意味
XmagicApi.XmagicYTDataListener ytDataListener	コールバック関数実装クラス。

表情
iOS

最終更新日：：2023-02-27 14:27:24

機能説明

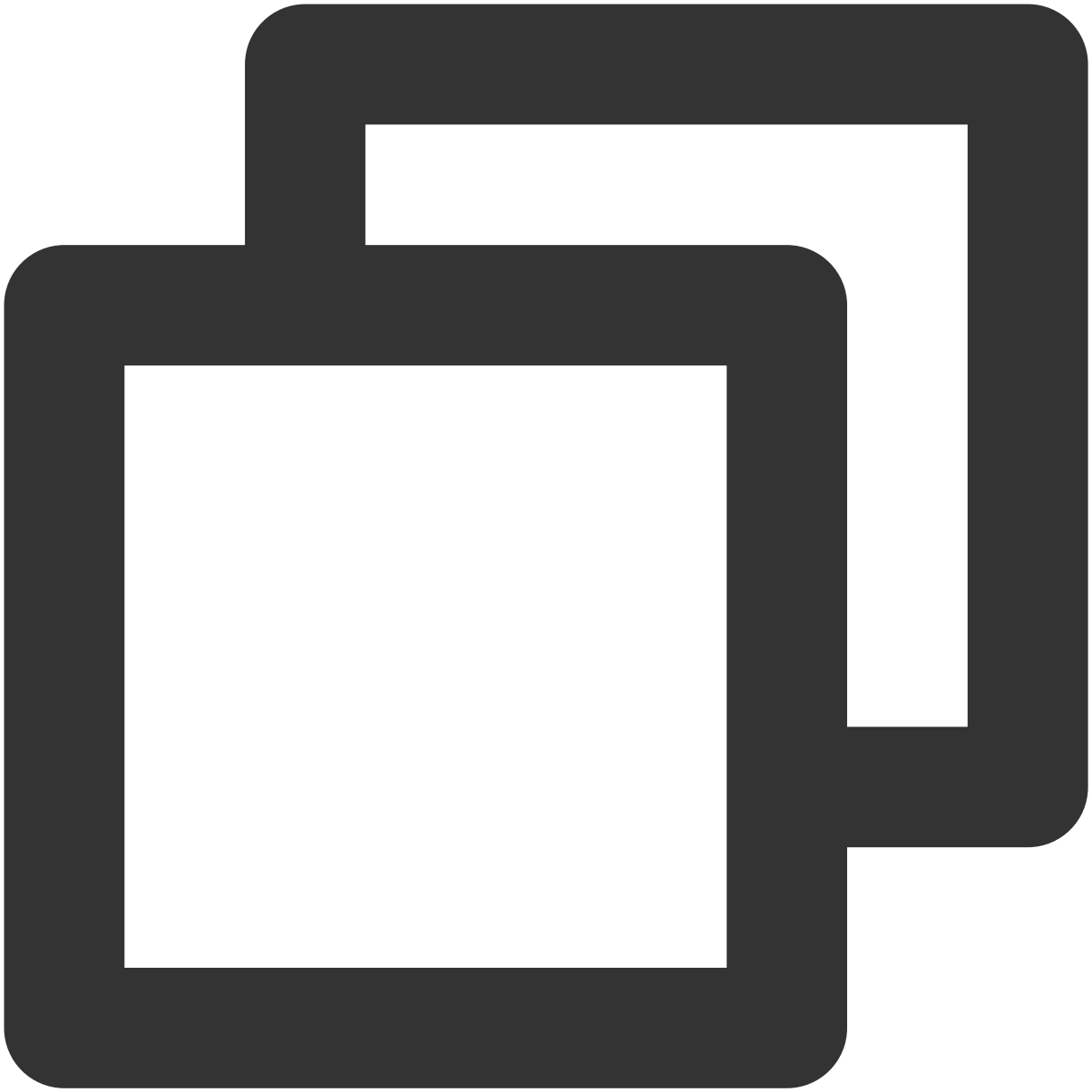
カメラのopenGLテクスチャを入力すると、Apple ARKitのルールに従って、52種類の顔の表情のBlendShapeデータがリアルタイムで出力されます。詳細については、ARFaceAnchor。これらの表情データを利用して、Unityに渡してモデルを動かすといった、さらに進んだ開発を行うことができます。

iOS統合ガイド

iOSのSDKの統合ガイドについての詳細は、Tencent Effectの独立した統合をご参照ください。

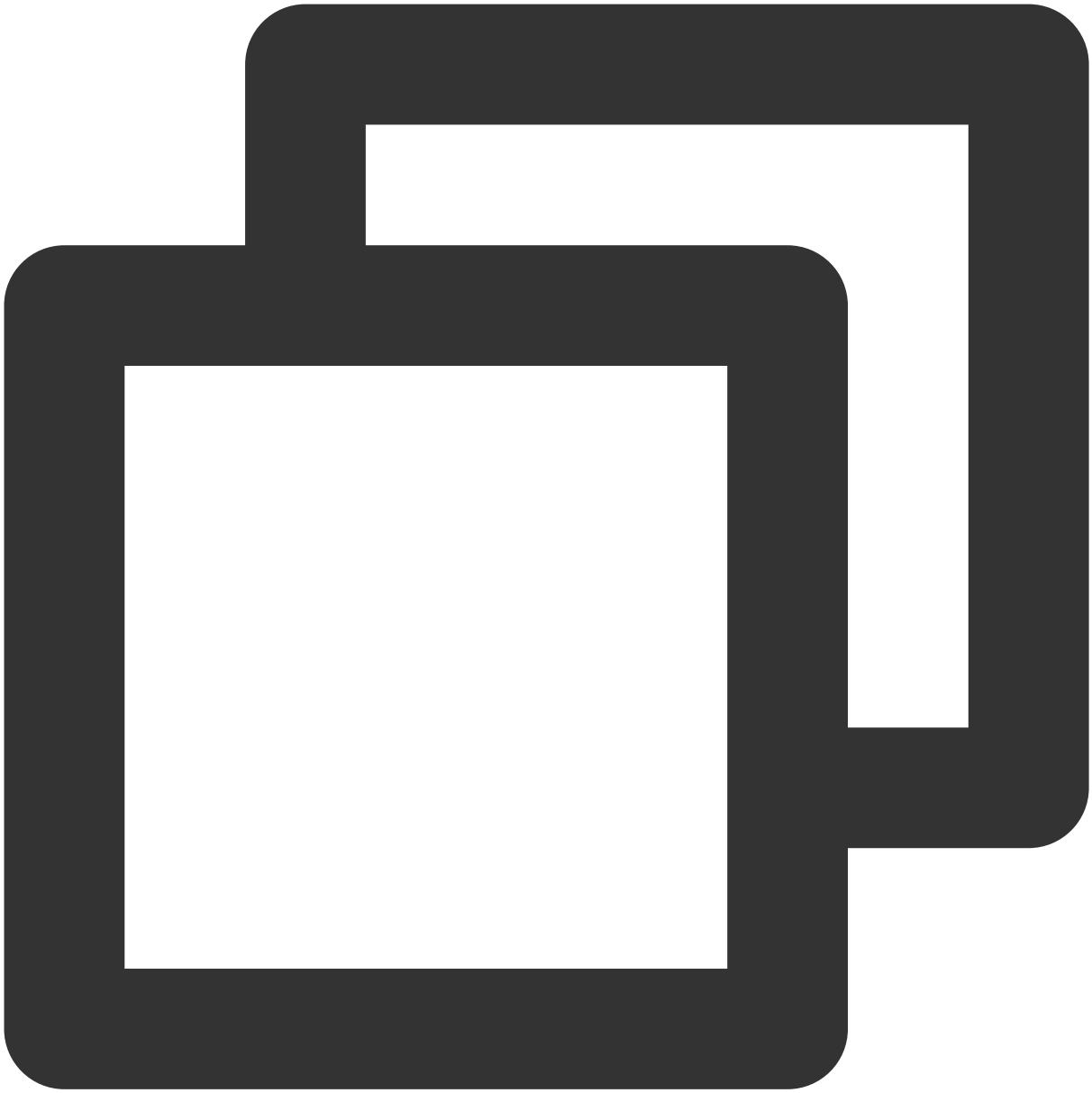
インターフェースの呼び出し

1. 機能スイッチをオンにします。



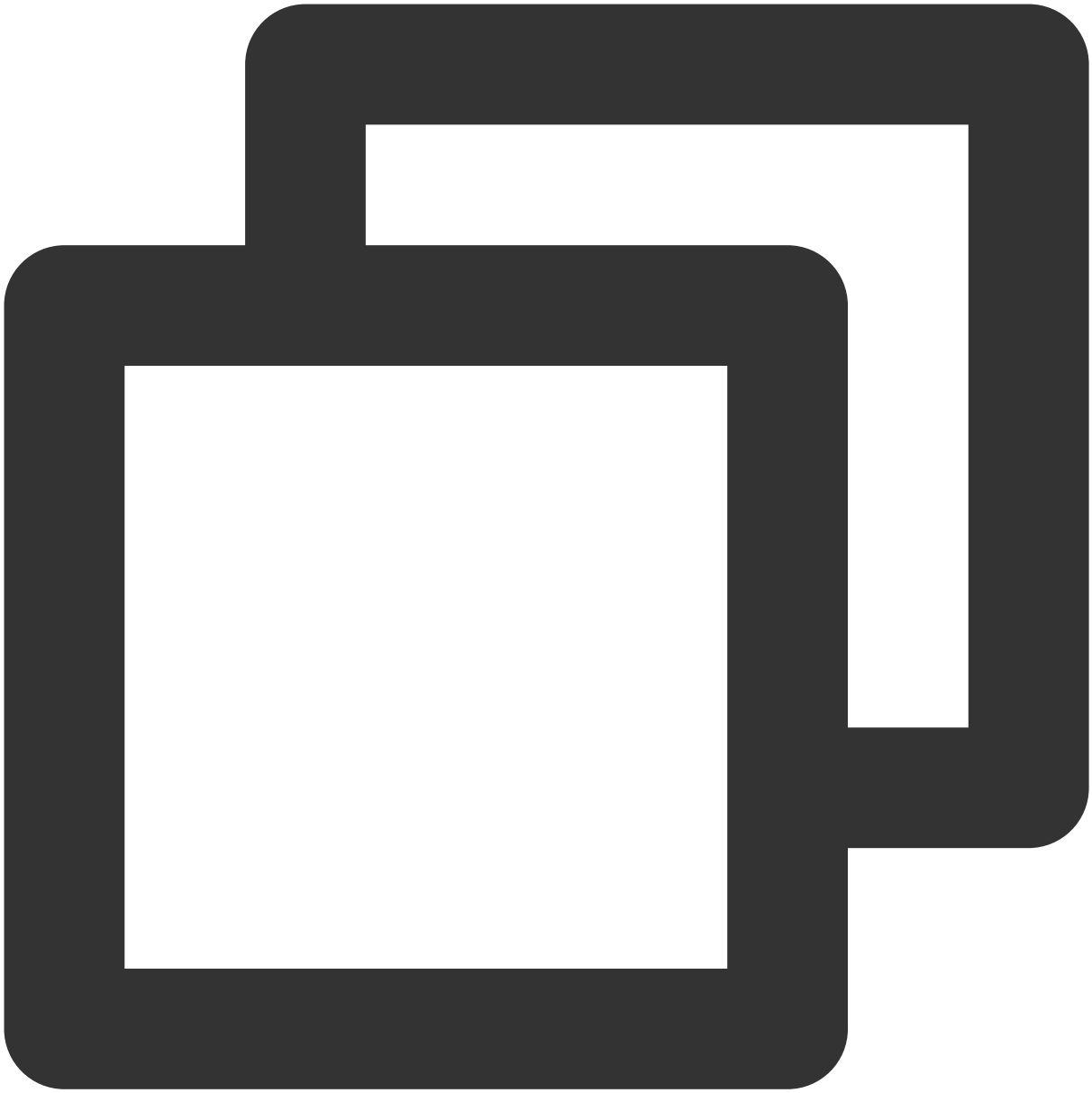
```
[self.beautyKit setFeatureEnableDisable:ANIMOJI_52_EXPRESSION enable:YES];
```

2. 顔の特徴点位置情報データのコールバックを設定します。



```
//XMagic.h
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

@implementation listener
- (void)onYTDataEvent:(id)event
{
    NSLog(@"YTData %@", event);
}
@end
```

onYTDataUpdateはJSON string構造を返します。最大で5つの顔の情報を返します。



```
{
 "face_info":[{
  "trace_id":5,
  "face_256_point":[
 180.0,
 112.2,
 ...
  ],
  "face_256_visible":[
 0.85,
 ...
```

```
    ],
    "out_of_screen":true,
    "left_eye_high_vis_ratio:1.0,
    "right_eye_high_vis_ratio":1.0,
    "left_eyebrow_high_vis_ratio":1.0,
    "right_eyebrow_high_vis_ratio":1.0,
    "mouth_high_vis_ratio":1.0,
    "expression_weights":[
       0.12,
       -0.32
       ...
   ]
   },
   ...
   ]
 }
```

フィールドの意味

trace_id：顔ID。連続ストリーム取得の過程で、IDが同一であれば同じ顔であると認識できます。
expression_weights：リアルタイムの表情blendshapeデータです。配列の長さは52で、各数値の値の範囲は0～1.0です。

{ "eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft","eyeSquintLeft","eyeWideLeft","eyeBlinkRight","eyeLookDownRight","eyeLookInRight","eyeLookOutRight","eyeLookUpRight","eyeSquintRight","eyeWideRight","jawForward","jawLeft","jawRight","jawOpen","mouthClose","mouthFunnel","mouthPucker","mouthRight","mouthLeft","mouthSmileLeft","mouthSmileRight","mouthFrownRight","mouthFrownLeft","mouthDimpleLeft","mouthDimpleRight","mouthStretchLeft","mouthStretchRight","mouthRollLower","mouthRollUpper","mouthShrugLower","mouthShrugUpper","mouthPressLeft","mouthPressRight","mouthLowerDownLeft","mouthLowerDownRight","mouthUpperUpLeft","mouthUpperUpRight","browDownLeft","browDownRight","browInnerUp","browOuterUpLeft","browOuterUpRight","cheekPuff","cheekSquintLeft","cheekSquintRight","noseSneerLeft","noseSneerRight","tongueOut"}

その他のフィールドは、顔情報にあります。関連するLicenseを購入した場合に、それらのフィールドを取得できます。表情データのみを取得したい場合は、それらのフィールドを無視してください。

Android

最終更新日：：2023-02-27 14:27:24

機能説明

カメラのopenGLテクスチャを入力すると、Apple ARKitのルールに従って、52種類の顔の表情のBlendShapeデータがリアルタイムで出力されます。詳細については、ARFaceAnchorをご参照ください。これらの表情データを利用して、Unityに渡してモデルを動かすといった、さらに進んだ開発を行うことができます。

Android統合ガイド

AndroidのSDKの統合ガイドについての詳細は、Tencent Effectの独立した統合をご参照ください。

インターフェースの呼び出し

1. 機能スイッチをオンにします。

```
//XmagicApi.java
//featureName = XmagicConstant.FeatureName.ANIMOJI_52_EXPRESSION
public void setFeatureEnableDisable(String featureName, boolean enable);
```

2. 顔の特徴点位置情報データのコールバックを設定します。

```
//XmagicApi.java
void setYTDataListener(XmagicApi.XmagicYTDataListener ytDataListener)
```

```
public interface XmagicYTDataListener {
void onYTDataUpdate(String data)
}
```

onYTDataUpdateはJSON string構造を返します。最大で5つの顔の情報を返します。

```
{
"face_info":[{
"trace_id":5,
"face_256_point":[
180.0,
112.2,
...
],
"face_256_visible":[
0.85,
...
],
"out_of_screen":true,
"left_eye_high_vis_ratio:1.0,
"right_eye_high_vis_ratio":1.0,
"left_eyebrow_high_vis_ratio":1.0,
"right_eyebrow_high_vis_ratio":1.0,
"mouth_high_vis_ratio":1.0,
"expression_weights":[
0.12,
-0.32
...
]
},
...
]
}
```

フィールドの意味

- **trace_id**：顔ID。連続ストリーム取得の過程で、IDが同一であれば同じ顔であると認識できます。
- **expression_weights**：リアルタイムの表情blendshapeデータです。配列の長さは52で、各数値の値の範囲は-1.0～1.0です。
- その他のフィールドは、顔情報にあります。関連するLicenseを購入した場合に、それらのフィールドを取得できます。表情データのみを取得したい場合は、それらのフィールドを無視してください。

体特徴点位置
iOS

最終更新日：：2023-03-06 10:07:51

機能説明

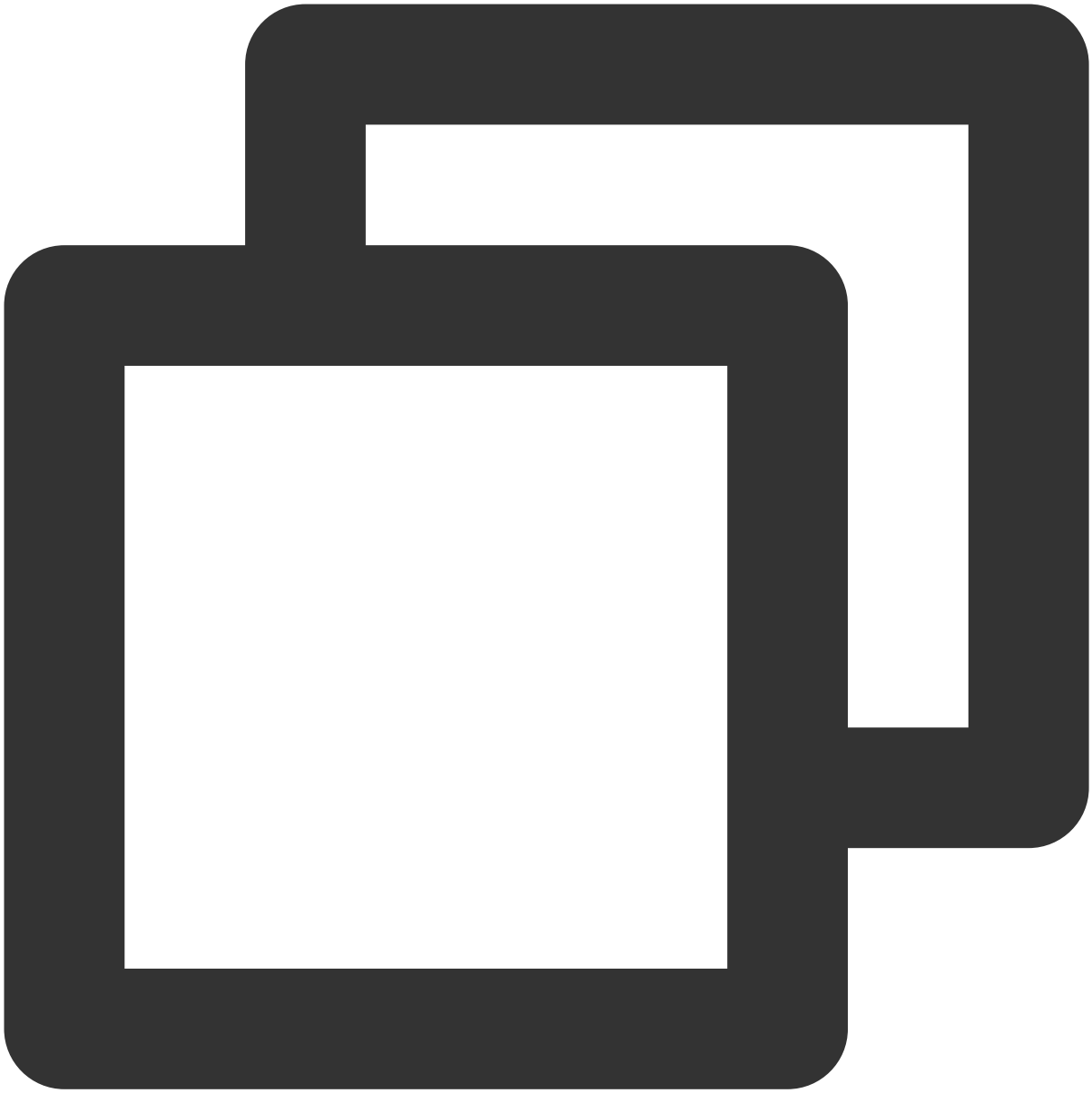
カメラのopenGLテクスチャを入力すると、体の3Dデータがリアルタイムで出力されます。これらの3Dデータを利用して、Unityに渡してモデルを動かすといった、さらに進んだ開発を行うことができます。

iOS統合ガイド

初めにTencent Effect SDKへの統合をする必要があります。具体的な内容は、Tencent Effectの独立した統合をご参照ください。

インターフェースの呼び出し

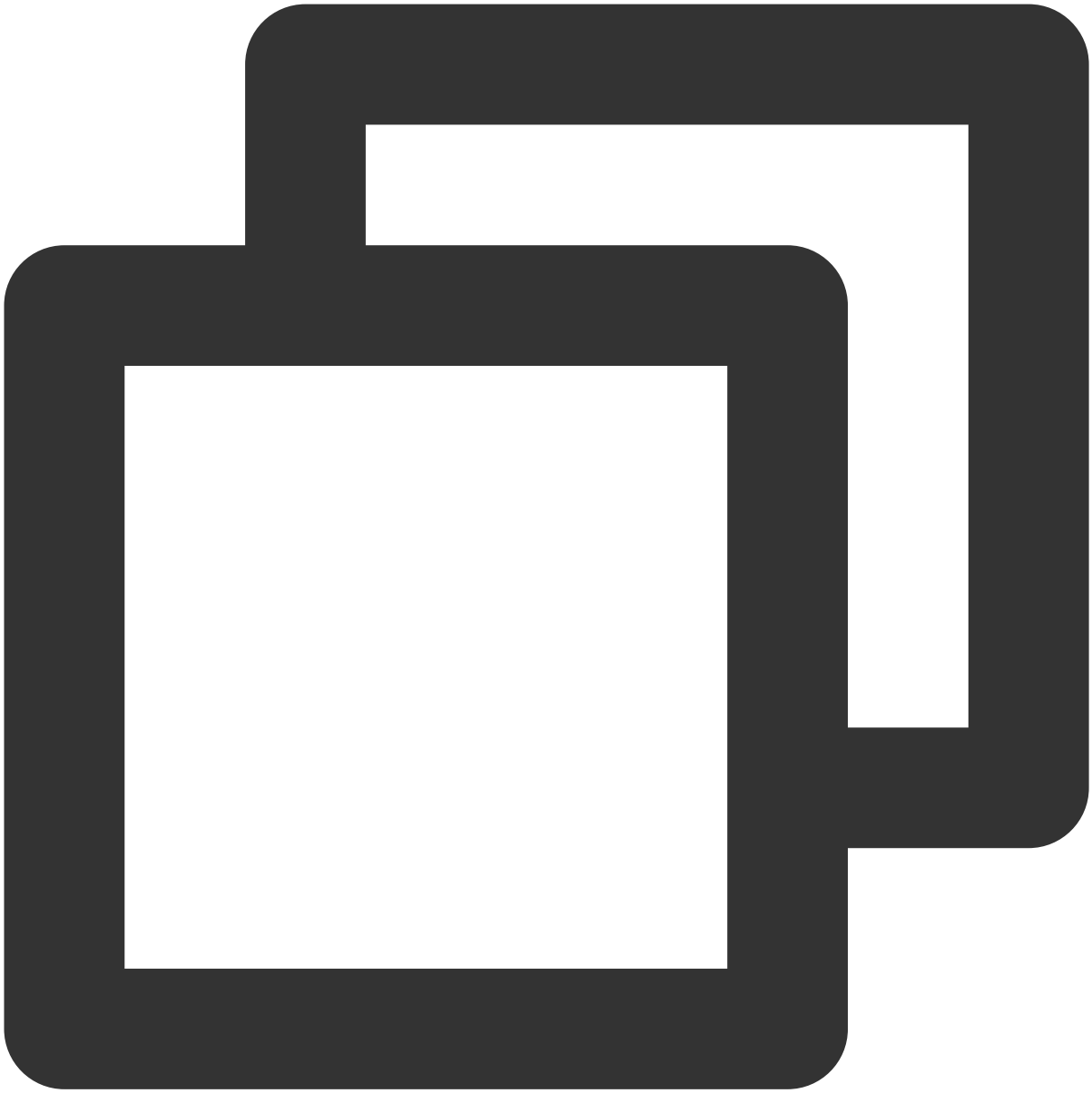
1. 機能スイッチをオンにします（XMagic.h）。



```
- (void)setFeatureEnableDisable:(NSString *_Nonnull)featureName enable:(BOOL)enable
```

featureNameに `XmagicConstant.FeatureName.BODY_3D_POINT` を入力します。

2. データコールバックの設定（XMagic.h）。



```
//XMagic.h
- (void)registerSDKEventListener:(id<YTSDKEventListener> _Nullable)listener;

@implementation listener
- (void)onYTDataEvent:(id)event
{
    NSLog(@"YTData %@", event);
}
@end
```

onYTDataEventはJSON構造のstringデータを返します。例：

「face_info」は、顔に関するデータです。体の3Dデータとは関係ありませんので、無視することができます。
「body_3d_info」の各フィールドの説明については、下記のとおりです

体の特徴点位置および特徴点位置データの説明

関連する説明については、体の特徴点位置およびデータの説明をご参照ください。

Android

最終更新日：：2023-02-27 14:27:24

機能説明

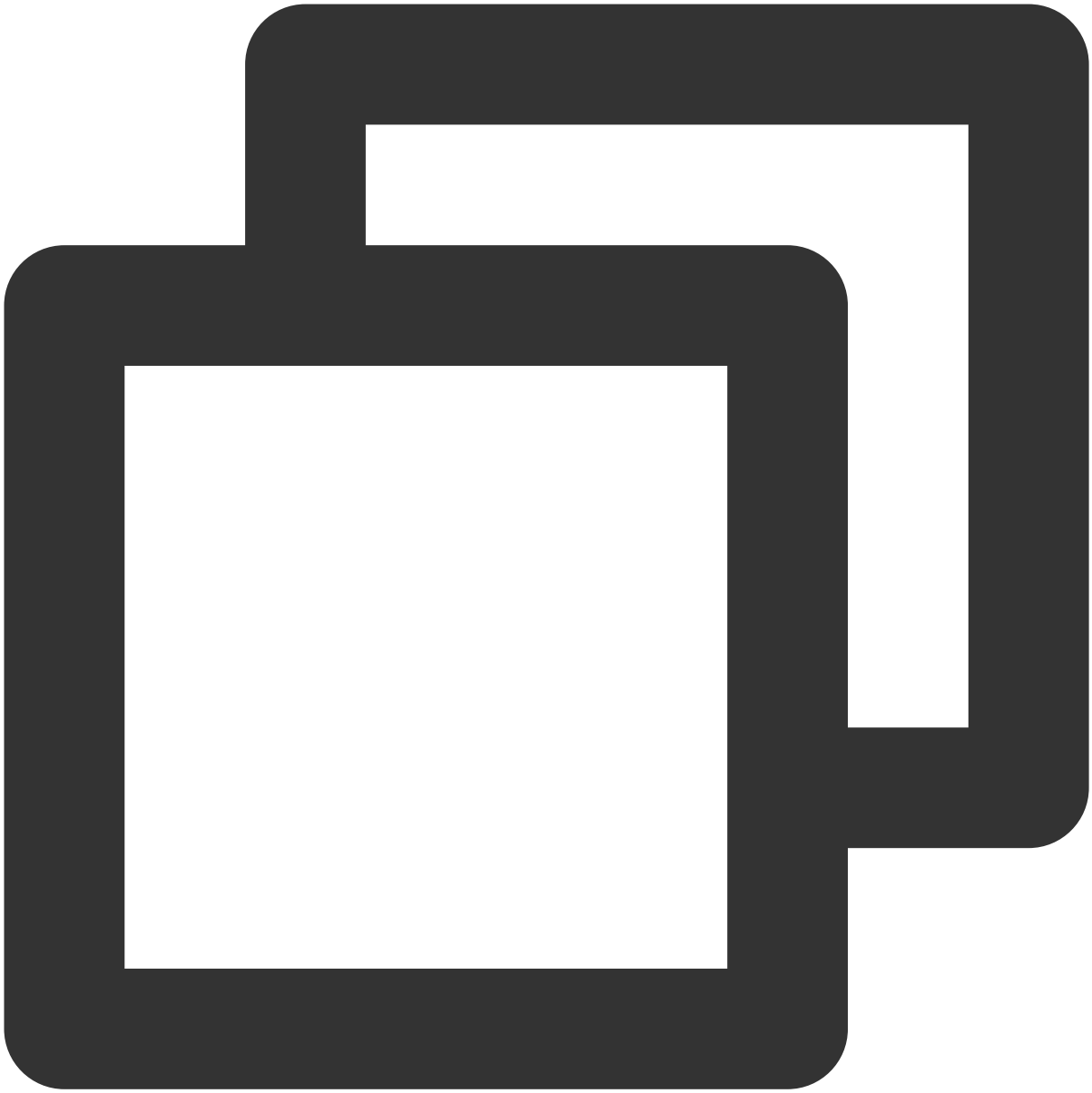
カメラのopenGLテクスチャを入力すると、体の3Dデータがリアルタイムで出力されます。これらの3Dデータを利用して、Unityに渡してモデルを動かすといった、さらに進んだ開発を行うことができます。

Android統合ガイド

初めにTencent Effect SDKへの統合をする必要があります。具体的な内容は、Tencent Effectの独立した統合をご参照ください。

インターフェースの呼び出し

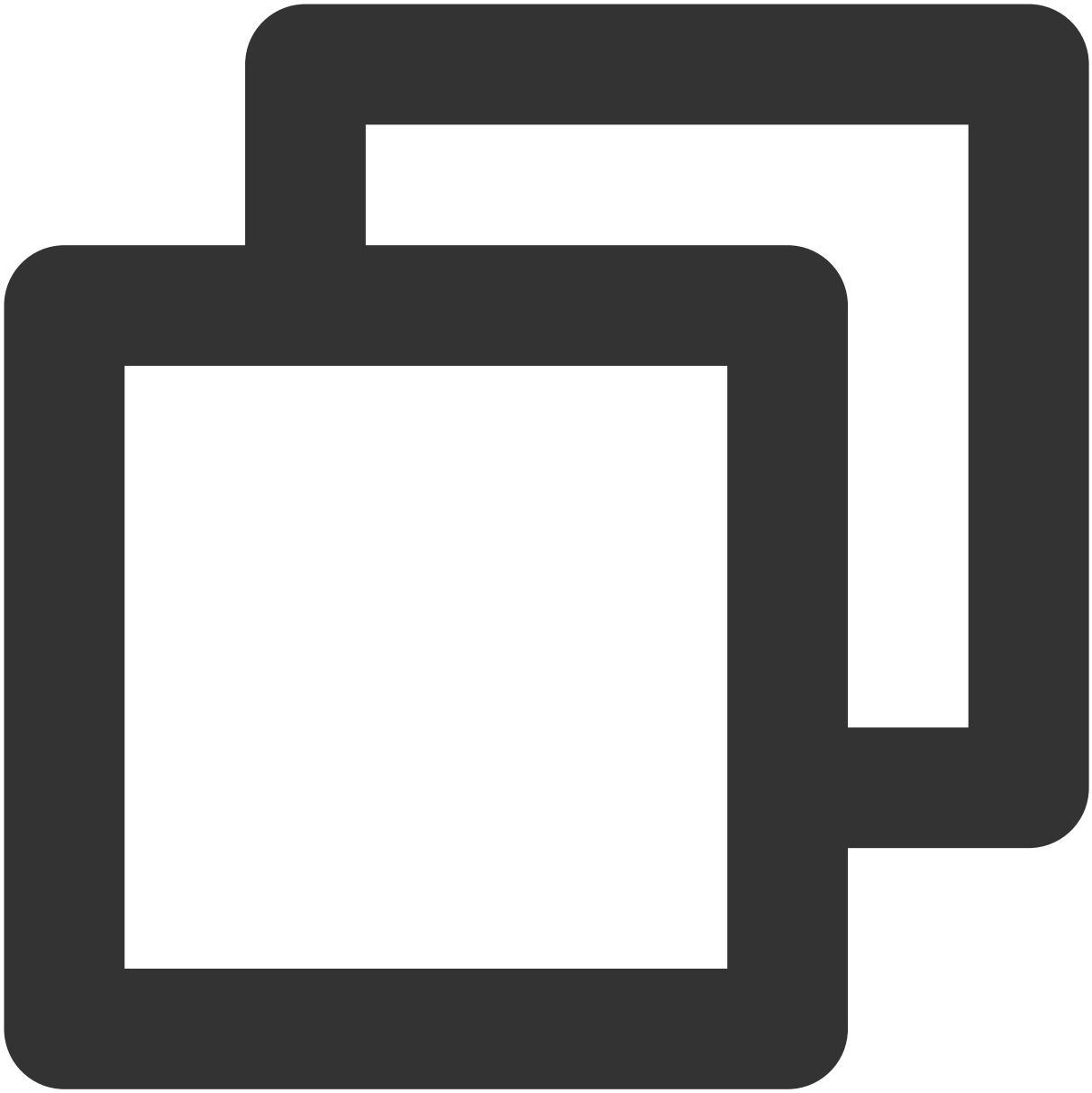
1. 機能スイッチをオンにします（XmagicApi.java）。



```
public void setFeatureEnableDisable(String featureName, boolean enable);
```

featureNameに `XmagicConstant.FeatureName.BODY_3D_POINT` を入力します。

2. データコールバックの設定（XmagicApi.java）。



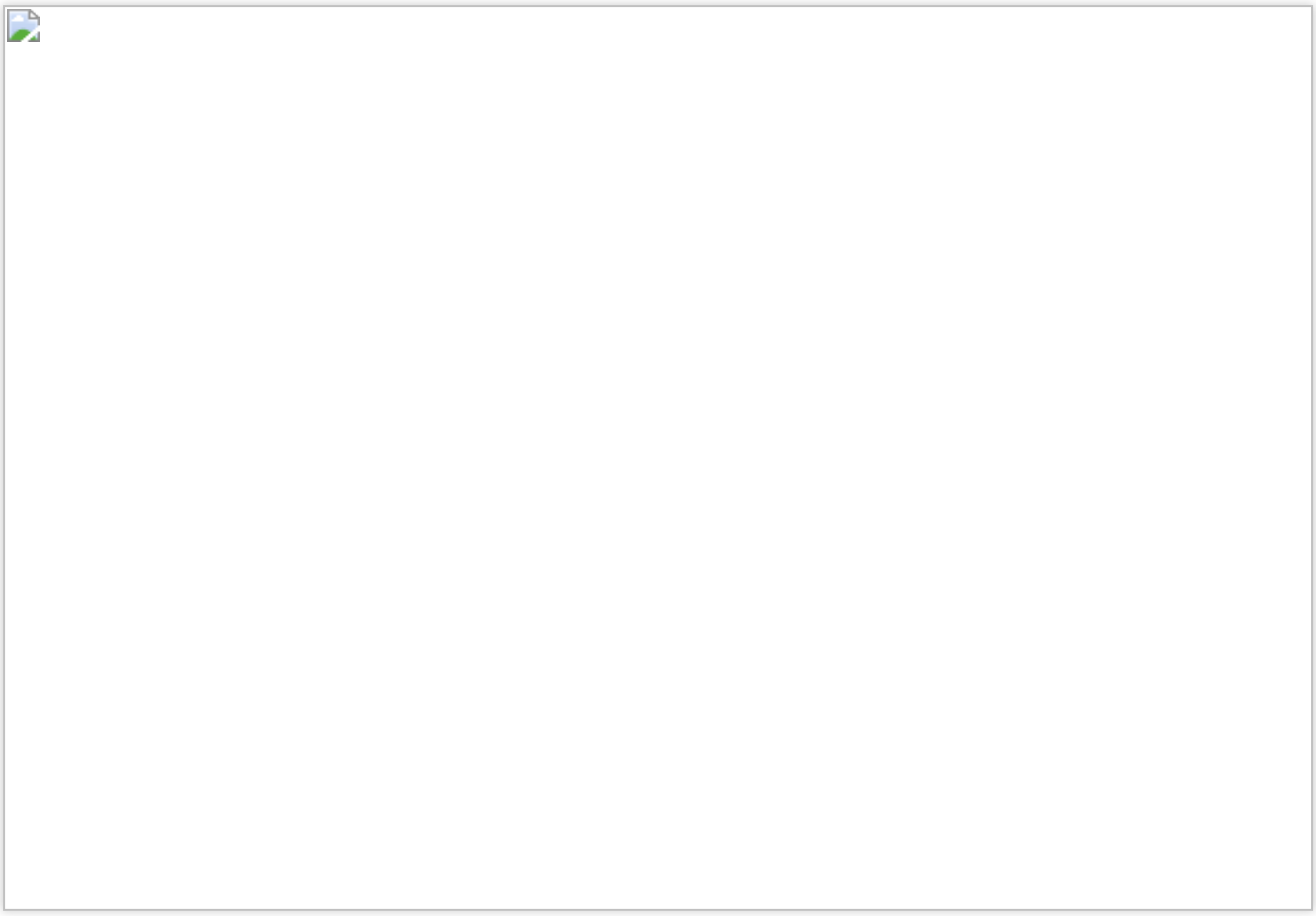
```
void setYTDataListener(XmagicApi.XmagicYTDataListener ytDataListener)

public interface XmagicYTDataListener {
 void onYTDataUpdate(String data)
}
```

onYTDataUpdateはJSON構造のstringデータを返します。例：
「face_info」は、顔に関するデータです。体の3Dデータとは関係ありませんので、無視することができます。
「body_3d_info」の各フィールドの説明については、下記のとおりです

体の特徴点位置および特徴点位置データの説明

標準SMPL特徴点位置の定義

標準SMPLX手の骨格の特徴点位置の定義



SDKが出力するJSONデータの例は、次のとおりです。

body_3d_infoの各フィールドの説明については、下記のとおりです。

imageWidth、imageHeight：SDKの画像の幅と高さを入力します

items：配列。現時点では1つの要素のみです

index：位置の保持。現時点では無視することができます

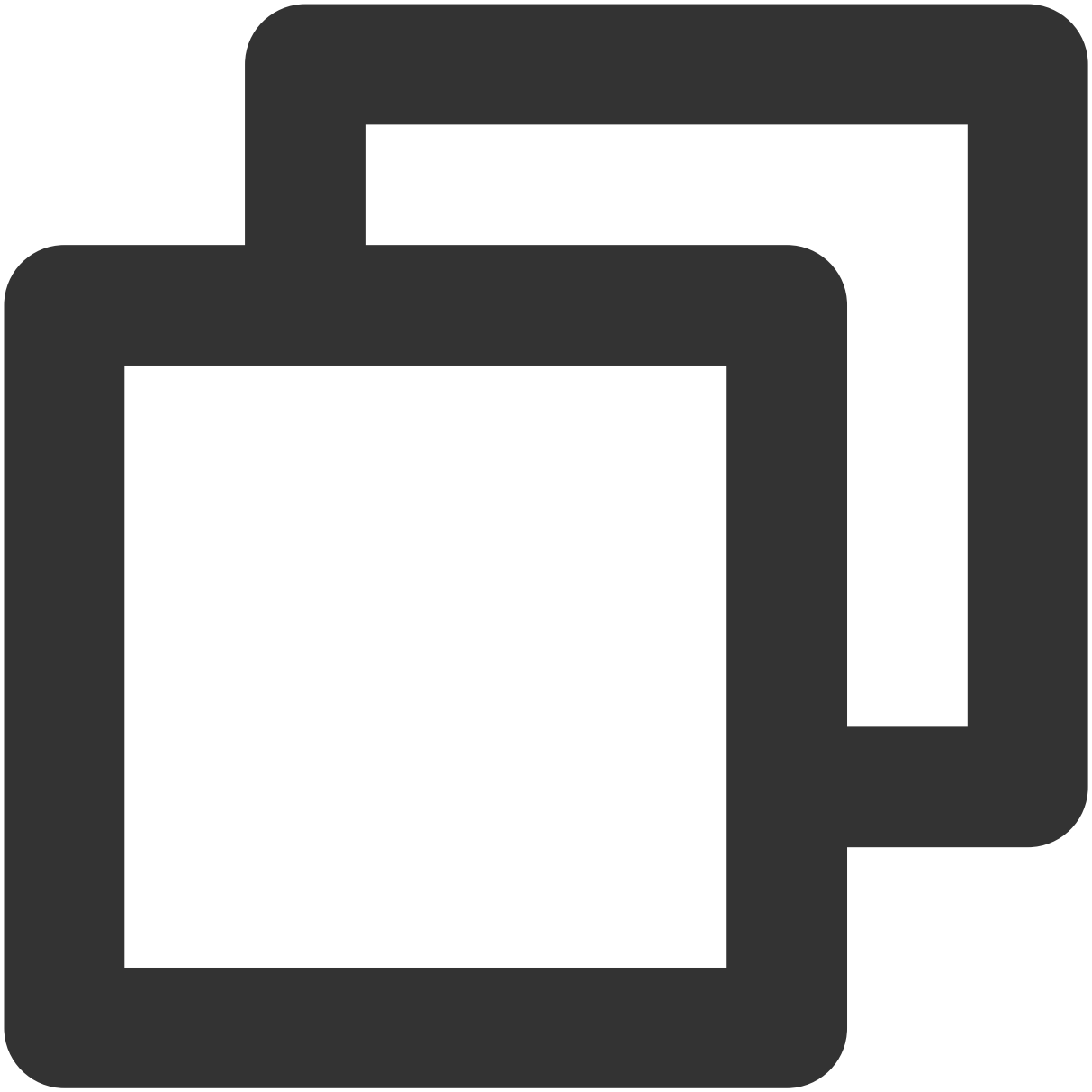
pose:

（1） [0,2]位置。人の位置はカメラ中心で、人の骨格の3D位置はxyzです

（2）[3,12]位置。人のタイプは10float数です。標準SMPLの10パッケージの異なるmeshをベースとした、人のタイプに対する総合的な評価を得ることができます

（3）[13]位置。Focal_lengthで、固定値は5000です

（4）[14,29]位置。OpenGL投影マトリックス、focal_lengthに基づいて取得する3D空間で、物体の投影マトリックスをレンダリングします。4X4の投影マトリックスのアルゴリズム内部における計算方法は、次のとおりです。



```
matrix={
    2 * focal_length / img_wid, 0, 0, 0,
    0, 2 * focal_length / img_hei, 0,0,
    0,0, (zf + zn) / (zn - zf), -1,
    0, 0, (2.0f * zf * zn) / (zn - zf), 0};
}
```

（5）[30,33]位置。接地データ、足が地面についているかどうか、左かかと、左つま先、右かかと、右つま先

position_x,position_y,position_z:

（1）[0,23]位置。人の2D特徴点位置です。上記の図1をご参照ください。2Dの点のposition_zはすべて0です

（2）[24,47]位置。人の3D特徴点位置です。上記の図1をご参照ください

rotation

（1）[0,23]位置。人の骨格の回転は4元数であり、各4元数の属性の順序はwxyzです

（2）[25,54]位置、手の骨格の回転は4元数です。左手が15で、右手も15で、各4元数の属性の順序はwxyzです

骨格の異なる命名方法および対応関係

番号	Bone Names	Bone Names 2
0	"pelvis",	"Hips"
1	"left_hip",	"LeftUpLeg"
2	"right_hip",	"RightUpLeg"
3	"spine1",	"Spine"
4	"left_knee",	"LeftLeg"
5	"right_knee",	"RightLeg"
6	"spine2",	"Spine1"
7	"left_ankle",	"LeftFoot"
8	"right_ankle",	"RightFoot"
9	"spine3",	"Spine2"
10	"left_foot",	""
11	"right_foot",	""
12	"neck",	"Neck"
13	"left_collar",	"LeftShoulder"
14	"right_collar",	"RightShoulder"
15	"head",	"Head"
16	"left_shoulder",	"LeftArm"
17	"right_shoulder",	"RightArm"
18	"left_elbow",	"LeftForeArm"
19	"right_elbow",	"RightForeArm"
20	"left_wrist",	"LeftHand"
21	"right_wrist",	"RightHand"
22	"left_hand"	""
23	"right_hand"	""
25	"left_index1"	IndexFinger1_L
26	"left_index2"	IndexFinger2_L
27	"left_index3"	IndexFinger3_L
28	"left_middle1"	MiddleFinger1_L
29	"left_middle2"	MiddleFinger2_L
30	"left_middle3"	MiddleFinger3_L
31	"left_pinky1"	PinkyFinger1_L
32	"left_pinky2"	PinkyFinger2_L
33	"left_pinky3"	PinkyFinger3_L

34	"left_ring1"	RingFinger1_L
35	"left_ring2"	RingFinger2_L
36	"left_ring3"	RingFinger3_L
37	"left_thumb1"	ThumbFinger1_L
38	"left_thumb2"	ThumbFinger2_L
39	"left_thumb3	ThumbFinger3_L
40	"right_index1"	IndexFinger1_R
41	"right_index2"	IndexFinger2_R
42	"right_index3"	IndexFinger3_R
43	"right_middle1"	MiddleFinger1_R
44	"right_middle2"	MiddleFinger2_R
45	"right_middle3"	MiddleFinger3_R
46	"right_pinky1"	PinkyFinger1_R
47	"right_pinky2"	PinkyFinger2_R
48	"right_pinky3"	PinkyFinger3_R
49	"right_ring1"	RingFinger1_R
50	"right_ring2"	RingFinger2_R
51	"right_ring3"	RingFinger3_R
52	"right_thumb1"	ThumbFinger1_R
53	"right_thumb2"	ThumbFinger2_R
54	"right_thumb3"	ThumbFinger3_R

音声表情変換
Android

最終更新日：：2023-02-27 12:19:12

機能説明

オーディオデータを入力すると、Apple ARKitの標準である52種類の表情のデータが出力されます。詳細については、ARFaceAnchorをご参照ください。これらの表情データを利用して、Unityに渡してモデルを動かすといった、さらに進んだ開発を行うことができます。

アクセス方法

方法1：Tencent Effect SDKによる接続

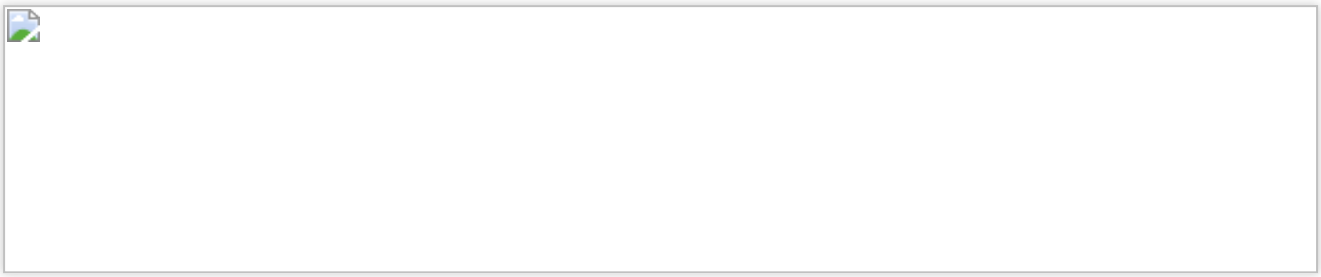
音声から表情への変換は、Tencent Effect SDKに統合されているため、最初の接続はTencent Effectのドキュメントに従って行う必要があります。
1. Tencent Effect SDK完全版をダウンロードします。
2. Tencent Effectの独立した統合 ドキュメントを参照して統合を完了します。

方法2：独立した音声表情変換SDKによる接続

音声から表情への変換のみが必要で、Tencent Effect SDKの機能を使用する必要がない場合は、独立した音声表情変換SDKの使用を検討することができます。aarパッケージは約6MBです。このSDKの取得については、アーキテクトまたは販売担当者までご連絡ください。

統合の手順

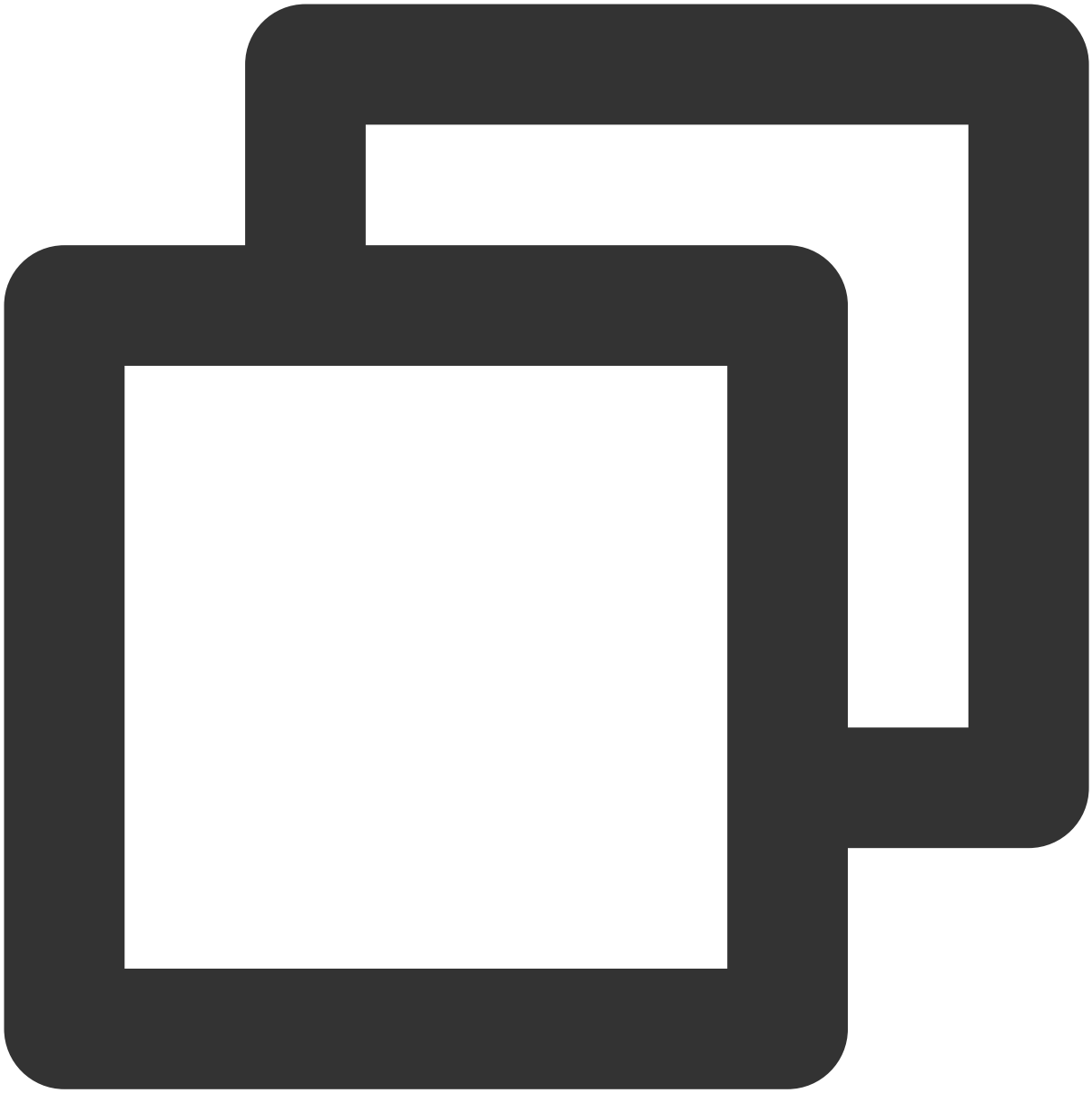
1. **Licenseを設定します**。認証をご参照ください。
2. **モデルファイルの設定**：必要なモデルファイルをassetsからappのプライベートディレクトリにコピーしてください（例：`context.getFilesDir() + "/my_models_dir/audio2exp"`。その後、Audio2ExpApiの`init(String modelPath)` インターフェースを呼び出した際、パラメータ `context.getFilesDir() + "/my_models_dir"` を渡します）。
モデルファイルはSDKパッケージにあります。場所は次のとおりです。



インターフェースの説明

インターフェース	説明
public int Audio2ExpApi.init(String modelPath);	初期化し、モデルのパスを渡します。上記の説明をご参照ください。戻り値
public float[] Audio2ExpApi.parseAudio(float[] inputData);	入力するのはオーディオデータであり、シングルチャネル、16Kサンプルレ順 `{"eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eye` す。
public int Audio2ExpApi.release();	使用完了後に呼び出し、リソースを解放します

統合コード例



```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.button).setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View view) {
                        TELicenseCheck.getInstance().setTELicense(MainActivity.this
                                @Override
                                public void onLicenseCheckFinish(int errorCode, Str
                                        Log.d(TAG, "onLicenseCheckFinish: errorCode
```

```
                                        if (errorCode == TELicenseCheck.ERROR_OK) {
                                                //license check success
                                                Audio2ExpApi audio2ExpApi = new Aud
                                                int err = audio2ExpApi.init(MainAct
                                                Log.d(TAG, "onLicenseCheckFinish: e
                                                //TODO start record and parse audio
                                        } else {
                                                // license check failed
                                        }
                                    }
                                });
                            }
                        });
    }
```

説明

完全なサンプルコードについては、美顔エフェクト SDK demoプロジェクトをご参照ください。

録音については、`com.tencent.demo.avatar.audio.AudioCapturer` をご参照ください。

インターフェースの使用については、`com.tencent.demo.avatar.activity.Audio2ExpActivity` およびその関連クラスをご参照ください。

iOS

最終更新日：：2023-02-27 12:16:36

機能説明

オーディオデータを入力すると、Apple ARKitの標準である52種類の表情のデータが出力されます。詳細については、ARFaceAnchorをご参照ください。これらの表情データを利用して、Unityに渡してモデルを動かすといった、さらに進んだ開発を行うことができます。

アクセス方法

方法1：Tencent Effect SDKと一緒に使用

1. 音声から表情sdkへの変換は、Tencent Effect SDKに結合されているため、最初の接続はTencent Effectのドキュメントに従って行う必要があります。
2. Tencent Effect SDK完全版をダウンロードします。
3. Tencent Effectの独立した統合のドキュメントを参照して統合を完了します。
4. Tencent Effect SDK完全版内のAudio2Exp.frameworkをプロジェクトに登録し、かつプロジェクトのtarget->General->Frameworks、Libraries、and Embedded ContentをEmbed & Signに設定します。

方法2：独立した音声表情変換SDKによる接続

音声から表情への変換のみが必要で、Tencent Effect SDKの機能を使用する必要がない場合は、独立した音声表情変換SDKの使用を検討することができます。Audio2Exp.frameworkパッケージは、約7MBです。プロジェクトにAudio2Exp.frameworkとYTCommonXMagic.frameworkの2つの動的ライブラリを導入し、かつプロジェクトのtarget->General->Frameworks、Libraries、and Embedded ContentをEmbed & Signに設定します。

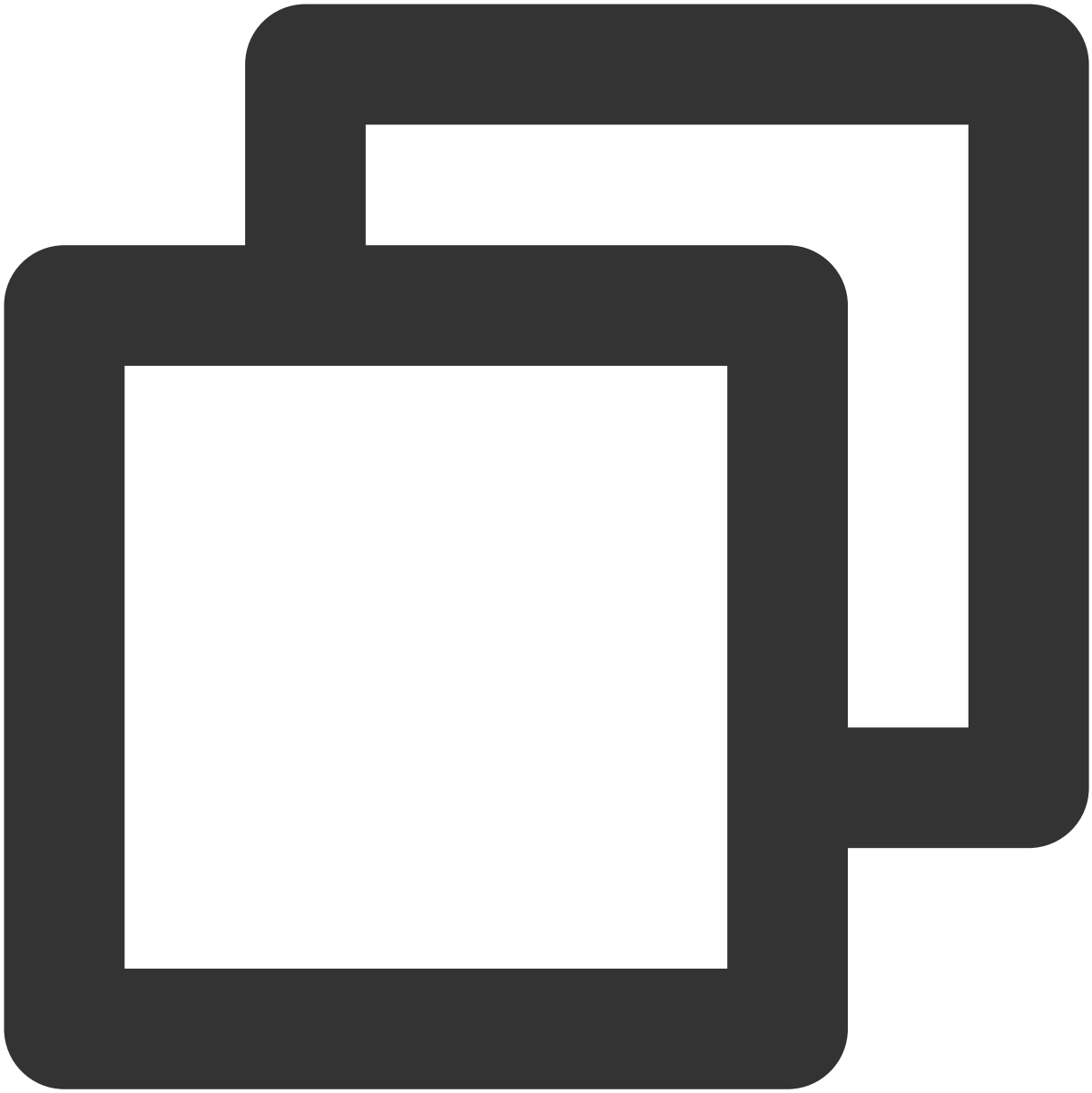
統合の手順

1. Licenseを設定します。認証 をご参照ください。
2. モデルファイルの設定：必要なモデルファイルaudio2exp.bundleをプロジェクトディレクトリにコピーします。その後、Audio2ExpApiのinitWithModelPath:インターフェースを呼び出した際、パラメータaudio2exp.bundle"をモデルファイルが存在するパスに渡します。

インターフェースの説明

インターフェース	説明
+ (int)initWithModelPath:(NSString*)modelPath;	初期化し、モデルのパスを渡します。上記の説明をご参照ください。戻り値が0の場合
+ (NSArray *)parseAudio:(NSArray *)inputData;	入力するのはオーディオデータであり、シングルチャネル、16Kサンプルレート、配列 {"eyeBlinkLeft","eyeLookDownLeft","eyeLookInLeft","eyeLookOutLeft","eyeLookUpLeft", す。
+ (int)releaseSdk	使用完了後に呼び出し、リソースを解放します

統合コード例



```
// 音声表情変換sdkの初期化
NSString *path = [[NSBundle mainBundle] pathForResource:@"audio2exp" ofType:@"bundl
int ret = [Audio2ExpApi initWithModelPath:path];
// 音声データから52種類の表情データへの変換
NSArray *emotionArray = [Audio2ExpApi parseAudio:floatArr];
// sdkのリリース
[Audio2ExpApi releaseSdk];

// Tencent Effect sdk xmgaicと結合して使用します
// 対応するリソースを使用して美顔sdkを初期化します
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize assetsDict:assetsDi
```

```
// avatar素材をロードします
[self.beautyKit loadAvatar:bundlePath exportedAvatar:nil completion:nil];
// 52の表情データを美顔sdkに渡すと、効果を確認することができます
[self.beautyKit updateAvatarByExpression:emotionArray];
```

説明：

完全なサンプルコードについては、美顔エフェクト SDK demoプロジェクトをご参照ください。

録音については、`TXCAudioRecorder` をご参照ください。

インターフェースの使用については、`VoiceViewController` およびその関連クラスをご参照ください。