Manual

SDK Android

Match Facial

Plataforma Android

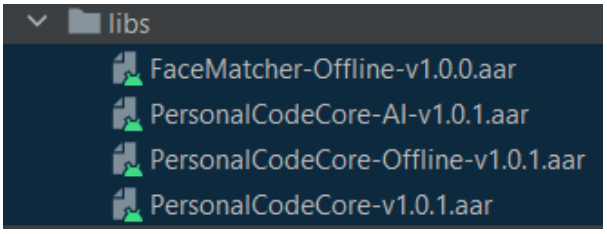
Versión 1.0.0

Manual de usuario para el programador Android de la librería Face Matcher Offline

1 - Importación del SDK al proyecto

1.1 - Copiar los siguientes archivos al proyecto dentro de la carpeta libs:

- PersonalCodeCore-v[VERSIÓN].aar
- PersonalCodeCore-AI-v[VERSIÓN].aar
- PersonalCodeCore-Offline-v[VERSIÓN].aar
- PersonalCodeMatcher-Offline-v[VERSIÓN].aar



1.2 - Agregar las líneas de implementación con sus respectivas versiones en el archivo build.gradle (Módulo app)

1.3 - Declarar las licencias referentes a Debug y Release en Gradle con las siguientes líneas de código:

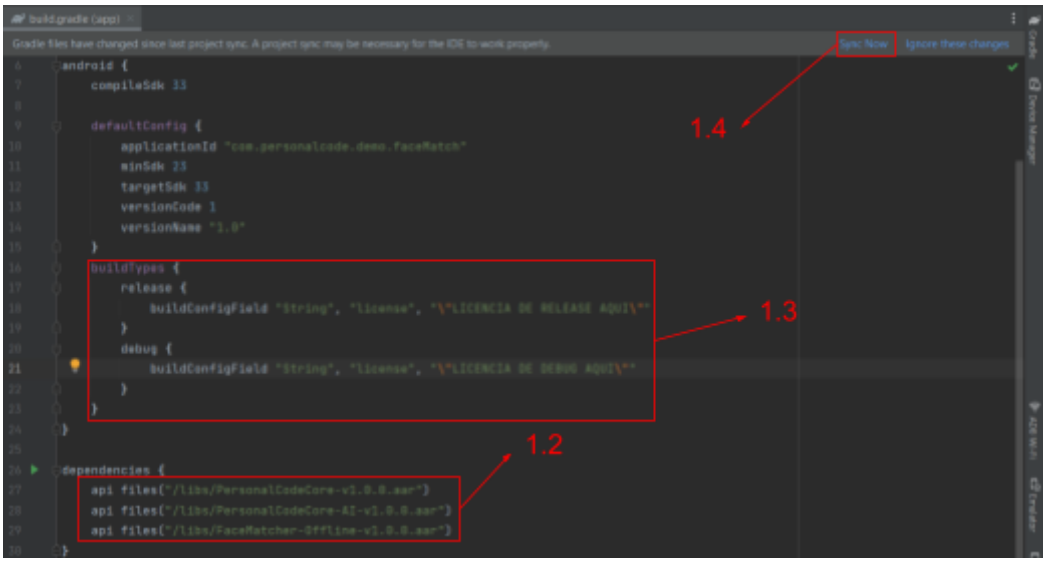
```
buildTypes {
    release {
        buildConfigField "String", "license", "\"Android|RELEASE_LICENSE\""
    }
    debug {
        buildConfigField "String", "license", "\"Android|DEBUG_LICENSE\""
    }
}
```

Las licencias que proveemos son firmadas y se compilan junto con la aplicación en su respectivo entorno. Es decir que una vez que se compila en modo Release, la apk no llevará la licencia de Debug y vice-versa. Al ser firmadas las licencias digitalmente, no se podrá cambiar su contenido, por lo que al necesitar algún cambio en la licencia se deberá solicitar dichos cambios directamente al proveedor.

Dichas licencias van vinculadas al applicationId y al SHA1 de compilación de la apk, por lo que es de extrema importancia que se resguarden las llaves de compilación

.jks, así como su keystore de Android Studio, de forma que solo los desarrolladores autorizados en las mismas puedan utilizar su licencia.

1.4 - Sincronizar el proyecto



2 - Implementación del SDK al proyecto

2.1 - Importar las siguientes clases a la clase que implementará el SDK:

```
import android.app.Activity
import android.graphics.Bitmap
import android.view.ViewGroup
import com.personalcode.FaceMatcherListener
import com.personalcode.PersonalCodeFaceMatcherOffline
```

2.2 - Inicializar una instancia de *FaceMatcherListener*, donde el SDK notificará sus eventos.

```
private var faceMatcherListener = object: FaceMatcherListener {
    override fun onFinished(activity: Activity, status: Int, description: String, similarity: Float) {
        //todo: Mostrar resultados de la comparación
    }

    override fun onOpenned(activity: Activity?, rootView: ViewGroup?) {
        //todo: Personalizar la vista aquí
    }

    override fun onCaptured(activity: Activity?, face2: Bitmap) {
        //todo: Mostrar rostro capturado
    }
}
```

2.3 - Inicializar el SDK mediante PersonalCodeFaceMatcherOffline.getInstance(), donde enviamos la instancia de *FaceMatcherListener* que se generó anteriormente.

```
init {
    faceMatcher = PersonalCodeFaceMatcherOffline.getInstance(activity, faceMatcherListener)
}
```

2.4 - Validar y solicitar permiso de cámara al usuario

3.0 - Métodos

3.1 - faceMatcher.capture(Activity activity, boolean finishLast):

Inicia el scanner facial para realizar una captura, donde:

- *Activity* es la instancia de la actividad actual de la aplicación
- Y el parámetro booleano solicita al SDK que cierre la actividad actual al abrir el scanner o no (true la cierra y false no hará nada)

3.2 - faceMatcher.captureAndMatch(Activity activity, boolean finishLast, Bitmap faceToMatch):

Inicia el scanner facial para realizar una captura y, al terminar la compara con el parámetro *faceToMatch* pasado al método.

- *Activity* es la instancia de la actividad actual de la aplicación
- El parámetro booleano solicita al SDK que cierre la actividad actual al abrir el scanner o no (true la cierra y false no hará nada)
- *faceToMatch* es una imagen *Bitmap* conteniendo un rostro para realizar la comparación.

3.3 - faceMatcher.matchFaces(Activity activity, Bitmap face1, Bitmap face2):

Realiza el match facial entre dos imágenes (parámetros face1 y face2).

- *Activity* es la instancia de la actividad actual de la aplicación
- *face1* es una imagen *Bitmap* conteniendo un rostro para realizar la comparación.
- *face2* es una imagen *Bitmap* conteniendo otro rostro para realizar la comparación.

4 - Personalización de la vista del Scanner

Para la personalización del scanner contamos con los siguientes atributos públicos en la clase *PersonalCodeFaceMatcherOffline* que pueden ser modificados según la necesidad al instanciar la clase (punto 2.3 del presente documento):

```
public int loadingGifResource = Integer.MIN_VALUE;
public String msgLoading = "Procesando...";
public String personalCodeColorLine = null;
public Typeface personalCodeFont = null;

public float defaultZoom = 1f;

public boolean changeFlashEnabled = true;
public boolean autoFlashEnabled = false;
public int defaultFlashMode = -1;
public Bitmap imgFlashOn = null;
public Bitmap imgFlashOff = null;
public Bitmap imgFlashAuto = null;

public int defaultCamera = 1;
public boolean changeCameraEnabled = true;
public Bitmap imgTurnCamera = null;

public int arrowColor = Integer.MIN_VALUE;
public Bitmap imgClose = null;
```

Donde:

- loadingGifResource: ID del recurso gráfico del gif de carga
- msgLoading: Mensaje a mostrar durante la carga
- personalCodeColorLine: Color del texto instructivo del scanner
- personalCodeFont: Fuente de los textos instructivos y de carga
- defaultZoom: Valor del Zoom de la cámara por defecto
- changeFlashEnabled: habilita o deshabilita el botón del flash
- autoFlashEnabled: habilita o deshabilita el flash automático
- defaultFlashMode: valor del flash por defecto (0 = apagado, 1 = prendido, 2 = automático)
- imgFlashOn: ícono del botón de flash prendido
- imgFlashOff: ícono del botón de flash apagado
- imgFlashAuto: ícono del botón de flash automático
- defaultCamera: cámara seleccionada por defecto (0 = trasera, 1 = frontal)
- changeCameraEnabled: habilita o deshabilita el botón de voltear la cámara
- imgTurnCamera: ícono del botón de voltear la cámara

- arrowColor: color de la flecha instructiva del scanner facial
- imgClose: ícono del botón de cierre del scanner

4.1 - Botones de Cierre, Flash, y voltear la cámara

Caso se necesite modificar la ubicación o el aspecto del botón de flash, se podrá obtener la instancia de la actividad *FMFaceCaptureActivity* mediante su método *FMFaceCaptureActivity.getInstance()* dentro del callback *onOpened()* de la interfaz *PersonalCodeInterface* creada en el punto 2.2 de éste documento, donde tiene el atributo público *btnFlash, btnClose o btnTurnCamera* del tipo *Button*.

Mediante la instancia de éste atributo será posible cambiar aspecto, tamaño y ubicación del botón.

4.2 - Otras personalizaciones

Para agregar logo y personalizar más a fondo la vista de *FMFaceCaptureActivity*, en el callback *onOpened()* de la interfaz *PersonalCodeInterface* se obtiene el parámetro *rootView* del tipo *ViewGroup*, donde se podrá implementar nuevas vistas con solo algunas pocas líneas de código.