VK CxHub Android SDK - документация по использованию для разработчика

VK CxHub SDK Android - это библиотека, позволяющая обрабатывать данные, отправленные из вашего кабинета CxHub, и показывать их в виде уведомлений на устройствах пользователей, а также собирать статистику взаимодействия пользователя с приложением.

В данной инструкции предполагается, что свое приложение у вас уже есть, вы зарегистрировали его:

Либо в Firebase (https://console.firebase.google.com/u/0/?pli=1)

Либо в Huawei Push Kit (https://developer.huawei.com/consumer/en/hms/huawei-pushkit/)

Либо в системе push-уведомлений RuStore (https://help.rustore.ru/rustore/for_developers/developer-documentation/sdk_push-notifications)

Интеграция CxHub SDK

Для интеграции нужно выполнить несколько шагов.

1) Добавить репозиторий в build.gradle (или в settings.gradle, начиная с версии gradle 6.8) в корне проекта - оттуда будет скачиваться VK CxHub SDK Android.

```
repositories {
    mavenCentral()
}
```

2) Добавить зависимость в build.gradle в модуле приложения.

```
implementation ("ru.mail.libnotify:cxhub-sdk:${cxhub_version}")
```

Здесь ${cxhub_version} - актуальная версия VK CxHub SDK Android. Сейчас это 0.1.0

3) Создать файл `cxhub.xml` в папке `res/values` со следующими полями:

- *cxhub_integration_id* - (string) Integration ID из кабинета CxHub
- *cxhub_application_secret* - (string) секретный ключ из кабинета CxHub
- *cxhub_resource_icon_id* - (путь к ресурсу drawable/mipmap) для иконки уведомлений. Нужно переопределить на иконку своего приложения (например, drawable/ic_message), иначе в уведомлениях будет показана иконка библиотеки по умолчанию
- *cxhub_api_host* - (string) путь к апи проекта. Например "https://test-project.apicxhub.ru/callback-service/"

Опционально можно добавить дополнительные поля (см. также Настройки ресурсов ниже):

- *cxhub_custom_font_** - Ресурсы шрифтов
- *cxhub_resource_sound* - (file) Звук уведомления
- *cxhub_resource_led_color_id* - (color) Цвет индикатора
- *cxhub_tracking_activity* - (true/false) Автоматическое слежение за жизненным циклом всех activity в приложении. false по умолчанию
- *cxhub_tracking_fragment* - (true/false) Автоматическое слежение за жизненным циклом всех fragment в приложении. false по умолчанию

При необходимости можно создать несколько конфигурационных файлов, например, для debug- и release-версии.

4) Начиная с Android 12, нужно добавить разрешение на отправку push-уведомлений в AndroidManifest.xml

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

5) Инициализировать и запустить VK CxHub SDK Android в onCreate методе приложения

```
public class App extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

         // Init what you use: FirebasePlatformManager, HuaweiPlatformManager or RuStorePlatformManager
         NotificationFactory.setPlatformManagers(
                FirebasePlatformManager.getInstance(),
                HuaweiPlatformManager.getInstance(),
                RustorePlatformManager.getInstance()
        );

        NotificationFactory.initialize(this);

        // Asynchrously starts CxHub SDK. You can call it later.
        // While it's not called SDK will not receive notifications
        NotificationFactory.bootstrap(this);
    }
}
```

PlatformManager определяет, через какую систему будут посылаться уведомления. Если указано несколько, приоритет отдается сначала Firebase, затем Huawei, затем RuStore, при условии, что соответствующие сервисы доступны на устройстве пользователя.

5) В зависимости от того, какой сервис вы используете, дополнительно в build.gradle в модуле приложения добавьте следующие зависимости:

```
# For Firebase Cloud Messaging
implementation ("ru.mail.libnotify:cxhub-firebase:${cxhub_version}")

# For Huawei Messaging Service
implementation ("ru.mail.libnotify:cxhub-huawei:${cxhub_version}")

 # ForRuStore Push SDK
implementation ("ru.mail.libnotify:cxhub-rustore:${cxhub_version}")
```

Также не забудьте добавить объявление для сервиса push-уведомлений в AndroidManifest.xml

```
<!-- For Firebase -->
<service
    android:name="cxhub.api.CxHubFirebaseMessagingService"
    tools:replace="android:exported,android:enabled"
    android:exported="true"
    android:enabled="true">
</service>


<!-- For Huawei -->
<service
    android:name="cxhub.api.CxHubHmsMessageService"
    tools:replace="android:exported,android:enabled"
    android:exported="true"
    android:enabled="true">
</service>

<!-- For RuStore -->
<service
    android:name="cxhub.api.RuStoreMessageHandlerService"
    tools:replace="android:exported,android:enabled"
    android:exported="true"
    android:enabled="true">
</service>
```

В примере в строке android:name указано имя сервиса из VK CxHub SDK Android. Вы можете использовать его, либо создать свой собственный сервис, если вы хотите обрабатывать входящие данные как-либо еще. В последнем случае нужно будет явно передать пришедшие данные из вашего сервиса в VK CxHub SDK Android.

Пример для Firebase:

```
public class GcmMessageHandlerService extends FirebaseMessagingService {
    @Override
    public void onMessageReceived(RemoteMessage message) {
        String from = message.getFrom();
        Map<String, String> data = message.getData();
        NotificationFactory.deliverPushMessageIntent(this, data);
    }

    @Override
    public void onNewToken(String token) {
        NotificationFactory.refreshPushToken(this);
    }
}
```

Пример для Huawei:

```
public class HmsMessageHandlerService extends HmsMessageService {
    @Override
    public void onMessageReceived(RemoteMessage message) {
        Map<String, String> data = message.getDataOfMap();
        NotificationFactory.deliverPushMessageIntent(this, data);
    }

    @Override
    public void onNewToken(String token) {
        NotificationFactory.refreshPushToken(this);
    }
}
```

Пример для RuStore:

```
class RuStorePushService : RuStoreMessagingService() {
    override fun onNewToken(token: String) {
        NotificationFactory.refreshPushToken(this)
    }

    override fun onMessageReceived(message: RemoteMessage) {
        val dataStr = message.data["data"]
        if (dataStr.isNullOrEmpty()) {
            return
        }
        val dataMap = JSONObject(dataStr).toMap()

        NotificationFactory.deliverPushMessageIntent(this, dataMap)
    }

    private fun JSONObject.toMap(): Map<String, String> =
        keys().asSequence().associateWith { this[it].toString() }

    override fun onError(errors: List<RuStorePushClientException>) {
        // Don't need to call SDK
    }

    override fun onDeletedMessages() {
        super.onDeletedMessages()
        // Don't need to call SDK
    }

}
```

Проверьте, что в модуле проекта добавлен файл:

Либо firebase google-services.json для Firebase: https://developers.google.com/android/guides/google-services-plugin?hl=ru)

Либо agconnect-services.json для Huawei: (https://developer.huawei.com/consumer/en/doc/development/quickApp-Guides/quickapp-get-agconnectfile-0000001117853750)

Либо добавлен ключ от RuStore в исходный код приложения.

На этом этапе интеграция завершена.

Не забудьте запросить разрешение на показ уведомлений в приложении.

Использование

С помощью VK CxHub SDK Android вы можете собирать информацию о взаимодействии пользователя с приложением, чтобы в дальнейшем использовать для создания кампаний в рассылках.

Список событий, которые библиотека собирает автоматически, добавлен в конце.

Все вызовы, описанные здесь, **опциональны.** Нужны ли они в вашем приложении, зависит от ваших целей.

1) Сохранить пользовательское свойство (в любой момент).

```
//     * @param key   event key
//     * @param value event value, may be only String
//     * @param @optional valueMap event subproperty value, may be only Map<String,String>
UserProperty property = new UserProperty("subscribed", "message", Collections.singletonMap("email", "true"))

NotificationFactory.get(this).setUserProperty(
            property,
            new UserPropertyApi.OnResultListener() {
                        @Override
                        public void onSuccess() {
                            Toast.makeText(context, "onSuccess", Toast.LENGTH_SHORT).show();
                        }

                        @Override
                        public void onFailed(Throwable exception) {
                            Toast.makeText(context, "onFailed: " + exception.getMessage(), Toast.LENGTH_SHORT).
show();
                        }
                    }
);
```

Свойства могут быть любые.

2) Передать данные пользователя при логине. Логин может быть по любому типу, определенному маркетологом в кабинете CxHub (например, телефон, email, номер карты лояльности и т.д.). Переданный тип логина обязательно должен быть заведен в кабинете!

```
NotificationFactory.get(this).setUserId("MyUserIdType", userId, true)
```

3) Отправить любые свои события из нужных мест приложения

```
//     * @param key   event key
//     * @param value event value, may be only String or Number
NotificationFactory.get(this).collectEvent("MyEventName", "MyEventValue");
```

4) Можно использовать автоматическое слежение за жизненным циклом всех activity (посылаются события Created/Started/Stopped) и fragment (посылаются события Attached/Detached) в приложении.

Для этого в ресурсах надо прописать

```
<string name= "cxhub_tracking_activity" > true </string>
<string name= "cxhub_tracking_fragment" > true </string>
```

Чтобы события отображались в статистике с желаемым названием, в классе, наследуемом от Application, нужно вызвать

```
NotificationFactory.get(this).setTrackingNameMap(mapOf(MainActivity::class.java to "MAIN_ACTIVITY"))
```

ProGuard не использует обфускацию для классов Activity и Fragment, такой трекинг будет работать корректно.

5) Посмотреть mobileInstance приложения, установленного на конкретном телефоне (например, чтобы отправить себе пуш при тестировании)

```
Log.i("MY", NotificationFactory.get(this).mobileInstance)
```

6) При необходимости можно установить listener для exceptions, необработанных библиотекой

```
NotificationFactory.setUncaughtExceptionListener { _, ex ->
    Log.e(LOG_TAG, "UncaughtExceptionListener", ex)
}
```

7) Так же при необходимости можно запросить pushToken или подписаться на него

```
NotificationApi.PushTokenListener listener = new NotificationApi.PushTokenListener() {
    @Override
    public void onReceived(@Nullable String pushToken) {
        Log.d(LOG_TAG, "Push token: $pushToken");
    }
};

// Receive once
NotificationFactory.get(this).getPushToken(listener);

// Subscribe to updates (first emit with current value immediately)
NotificationFactory.get(this).subscribeToPushToken(listener);

// Don't forget to unsubcribe
NotificationFactory.get(this).unsubscribeToPushToken(listener);
```

Настройки ресурсов (опционально)

В cxhub.xml можно определить следующие параметры для показа пушей.

Для настроек темной и светлой темы

Для светлой темы:

```
<string name="cxhub_resource_color_id" translatable="false">color/cxhub_dark_color_accent</string>
<string name="cxhub_resource_close_color_id" translatable="false">android:color/white</string>
<string name="cxhub_resource_bg_color_id" translatable="false">android:color/white</string>
<string name="cxhub_resource_text_color_id" translatable="false">android:color/black</string>
<string name="cxhub_resource_button_color_id" translatable="false">android:color/white</string>
```

Для темной темы:

```
<string name="cxhub_resource_dark_color_id" translatable="false">color/cxhub_color_accent</string>
<string name="cxhub_resource_dark_close_color_id" translatable="false">android:color/white</string>
<string name="cxhub_resource_dark_bg_color_id" translatable="false">android:color/black</string>
<string name="cxhub_resource_dark_text_color_id" translatable="false">android:color/white</string>
<string name="cxhub_resource_dark_button_color_id" translatable="false">android:color/white</string>
```

Можно также сменить используемый шрифт:

```
<string name="cxhub_custom_font_desc" translatable="false">fonts/Courgette.ttf</string>
<string name="cxhub_custom_font_title" translatable="false">fonts/SpicyRice.ttf</string>
<string name="cxhub_custom_font_button" translatable="false">fonts/Courgette.ttf</string>
```

Чтобы использовать разные шрифты для разных тем, можно добавить префиксы для ресурсов.

Трекинг активити/фрагментов:

```
<string name="cxhub_tracking_activity">[true/false(default)]</string>
<string name="cxhub_tracking_fragment">[true/false(default)]</string>
```

При значении "true" отправляются кастомные (MobileCustom) события CxActivityStarted/CxActivityStopped/CxFragmentStarted/CxFragmentStopped.

Карта имен для этих событий может быть задана методом NotificationApi.setTrackingMap(Map<Class, String>), не заданные типы отравляются в виде имен классов.

Список событий, автоматически посылаемых библиотекой

EVENT	VALUES	Доп. инфа	Когда посылается
CxAppFirstLaunch			При первом запуске приложения
CxAppOpen	Значение deeplink (String), если открыто по нему		При открытии приложения При вызове приложения из свернутого состояния При выходе из сна, если приложение открыто
CxInstallReferrer	Параметры ссылки, по которой произошла установка (String)		При первом запуске приложения, если установка произошла из Google Play или Huawei App Gallery, куда юзер попал по ссылке для установки

CxPushStatus	Delivered
	Dismissed
	Opened
	TtlExpired
	Duplicate
	BlockedByUser
		Есть доп. поле
		
		`"metadata": {`
		`"scenario_name":"bac",`
		`"scenario_uuid":"123",`
		`"step_name":"abc",`
		`"step_uuid":"456",`
		`"push_id":"28862"`
		`}`
		
		Properties для всех PushStatus
		
		`properties: {`
		`"landing_type":"Main / Deeplink / Weblink"`
		`}`
			При изменения состояния пуша.
			
			Delivered - успешная доставка пуша (если уведомления разрешены, показался значок)
			
			Dismissed - пуш смахнули
			
			Opened - юзер кликнул по пушу (по основной части или по кнопке)
			
			TtlExpired - закончился срок Ttl
			
			Duplicate - одинаковый пуш пришел два раза (не показываем)
			
			BlockedByUser - пользователь запретил показ пушей от приложения, либо запретил пуши для конкретного канала.

CxPushTokenChanged	`Drop`
	`New`
	`Change`
			При изменении Push Token для FCM, HMS или RuStore
			
			Drop - токен на устройстве сейчас пустой
			
			New - пришел токен от системы, до этого токена не было
			
			Change - пришел токен от системы, до этого был старый

CxError	CxPushServices NotAvailable CxGcmTokenFail ure FormatError InstanceNotMatc hed JsonError ContentUrlError GeneralError	Может быть поле metadata, если ошибка связана с пушом. ContentUrlError содержит properties с {"url" : < проблемная ссылка>"}, если мы ее знаем. Остальные ошибки могут содержать properties с description: <текст ошибки>	CxPushServicesNotAvailable - При ошибке получения пуш токена от сервиса CxGcmTokenFailure - Если пуш-сервис (Google, Huawei, RuStore) не установлен на устройстве FormatError - формат json неправильный InstanceNotMatched - пуш пришел на неправильный инстанс или приложение JsonError - не смогли распарсить json ContentUrlError - не смогли открыть такую web- или deeplink страницу/ссылку GeneralError - все остальные ошибки. Получили пуш, и что-то еще пошло не так