Thunder Smart SDK مستند

(Android)

تاریخ	نسخه مستند	نسخه SDK
1403/04/10	1.0	1.2_5
1403/07/21	1.1	1.2_5_rev1
1403/07/25	1.2	1.2_5_rev1

مستند SDK

مقدمه

این کلاس شامل متدهای مختلفی است که برای انجام وظایف خاصی مانند انجام تراکنش‌های خرید و خدماتی، استعلام موجودی حساب، تأیید تراکنش‌ها، اصلاحیه تراکنش‌ها و چاپ رسید طراحی شده‌اند. هر متد به‌گونه‌ای طراحی شده است که با جنبه‌های مختلف SDK تعامل داشته باشد و در چارچوب یک Android Activity مورد استفاده قرار گیرد. همچنین این کلاس ارتباط با callbackها را مدیریت می‌کند که بازخوردی از موفقیت یا شکست عملیات ارائه می‌دهند و تضمین می‌کنند که توسعه‌دهندگان می‌توانند به راحتی این ویژگی‌ها را در برنامه‌های خود ادغام کنند.

کلاس GeneralSDKManager یک کلاس نهایی (final) است، به این معنا که نمی‌توان آن را subclass کرد و این امر تضمین می‌کند که عملکرد آن ثابت و ایمن باقی بماند. این کلاس همچنین دارای یک مرجع به HostApp از طریق کلاس HostApp است که با فراخوانی متد ()init مقداردهی اولیه می‌شود. این مقداردهی اولیه حیاتی است زیرا محیط لازم برای عملکرد مؤثر SDK در برنامه میزبان را فراهم می‌کند.

با استفاده از متدهای ارائه شده توسط GeneralSDKManager، توسعه‌دهندگان می‌توانند به طور یکپارچه Thunder Smart SDK را در برنامه‌های خود ادغام کنند و قابلیت‌های متنوعی مرتبط با تراکنش‌ها را فراهم آورند.

پیش نیاز

برای اجرای کدها و مثال ها نیاز به آخرین نسخه از **Android Studio** می باشد.

نرم افزار Thunder Smart SDK

Thunder Smart SDK که از طریق کلاس GeneralSDKManager قابل دسترسی است، برای تسهیل پردازش امن و کارآمد تراکنش‌ها در برنامه‌های اندرویدی طراحی شده است. این SDK می‌تواند برای فعالیت‌های مختلف مرتبط با پرداخت، از جمله تراکنش‌های خرید، استعلام موجودی و تراکنش‌های خدماتی استفاده شود. با عمل به عنوان یک برنامه third-party، Thunder Smart SDK این امکان را فراهم می‌کند که کاربران پرداخت‌های موبایلی خود را به صورت امن و یکپارچه انجام دهند و به طور مستقیم با برنامه میزبان ادغام شود.

فرآیند متد

مقداردهی اولیه

قبل از اینکه هر تراکنشی پردازش شود، باید کلاس GeneralSDKManager با استفاده از متد ()init مقداردهی اولیه شود. این متد تنظیمات لازم را از برنامه میزبان دریافت کرده و SDK را برای عملیات‌های بعدی آماده می‌کند.

درخواست تراکنش مالی

برای انجام یک تراکنش، مانند خرید یا خدمات، متد مرتبط مثلاً doSaleTransaction یا doServiceTransaction فراخوانی می‌شود. این متد نیاز به پارامترهای خاصی دارد، از جمله جزئیات تراکنش (مانند مبلغ یا نوع درخواست) و یک شیءcallback که نتیجه تراکنش را مدیریت می‌کند.

پردازش

پس از فراخوانی متدها، SDK درخواست را پردازش می‌کند. بسته به نوع درخواست، SDK یا عملیات را بلافاصله تکمیل می‌کند یا نیاز به اقدامات بیشتر، مانند تأیید کاربر یا تعامل با سیستم‌های خارجی دارد.

مدیریت Callback

پس از پردازش تراکنش، نتیجه از طریق رابط callback ارائه شده (TransactionCallBack یا ResultCallBack) برگردانده می‌شود. متدهای callback جزئیات مربوط به موفقیت یا خطاهای احتمالی تراکنش را ارائه می‌دهند.

نکات مهم

پیاده‌سازیCallback : پیاده‌سازی صحیح رابط‌های callback ضروری است. اگر callback های SDK به درستی مدیریت نشوند، ممکن است تراکنش به درستی به پایان نرسد و دستگاه قادر به انجام تراکنش‌های بعدی نباشد.

اصلاحیه تراکنش: در برخی پیکربندی‌ها، SDK ممکن است اجازه اصلاحیه تراکنش را بر اساس نوع درخواست و تنظیمات متد بدهد یا ندهد. توسعه‌دهندگان باید از این تنظیمات آگاه باشند تا رفتار مورد نظر را در برنامه‌های خود پیاده‌سازی کنند.

فعال سازی SDK

برای استفاده از این سرویس باید مراحل زیر پیاده سازی شوند:

مرحله ۱: کپی کردن فایل Smart-Light_SDK.aar

ابتدا، فایل Smart-Light_SDK*.aar را در پروژه خود کپی کنید. این فایل باید در پوشه libs پروژه اندرویدی شما قرار گیرد. پوشه libs معمولاً در سطح ریشه ماژول شما قرار دارد.

مرحله ۲: افزودن وابستگی به `build.gradle`

در مرحله بعد، باید وابستگی SDK را در فایل build.gradle در سطح ماژول خود اعلام کنید. این کار با استفاده از دستور implementation انجام می‌شود که همه فایل‌های .jar و .aar موجود در پوشه libs را شامل می‌شود.

این خط به Gradle می‌گوید که همه فایل‌های .jar و .aar موجود در پوشه libs را به عنوان وابستگی‌های پروژه شامل کند.

```
dependencies {
    implementation fileTree(include: ['*.jar', '*.aar'], dir: 'libs')
}
```

پس از انجام این تغییرات، پروژه خود را با Gradle همگام‌سازی کنید. این کار را می‌توانید با کلیک روی "Sync Now" که ظاهر می‌شود یا با استفاده از گزینه "Sync Project with Gradle Files" در Android Studio انجام دهید.

مرحله ۳: مقداردهی اولیه SDK در کد

پس از اضافه کردن فایل SDK و تنظیم وابستگی‌ها، مرحله بعدی مقداردهی اولیه SDK در کد است. این کار باید در مکانی مناسب در برنامه شما انجام شود، مانند Activity یا Fragment.

```
class MyFragment : Fragment() {

    var sdkManager: GeneralSDKManagerInterface = GeneralSDKManager.getInstance()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        sdkManager.init(requireActivity())
        Log.d("MyFragment", "SDK Manager initialized")
    }
}
```

متد ()init روی نمونه GeneralSDKManager فراخوانی می‌شود و ()requireActivity به عنوان context ارسال می‌شود. این context برای مقداردهی اولیه SDK ضروری است.

متد init

متد init برای مقداردهی اولیه Thunder Smart SDK استفاده می‌شود و تنظیمات لازم را برای عملکرد صحیح SDK در برنامه میزبان انجام می‌دهد.

پارامترهای ورودی init

عنوان	نوع داده	کاربرد
Context	ActivityContext	برای مقداردهی اولیه SDK استفاده می‌شود.

نحوه استفاده از init

```
sdkManager.init(requireActivity())
```

انجام تراکنش خرید

متد doSaleTransaction برای انجام یک تراکنش خرید با مبلغ مشخص و شماره فاکتور استفاده می‌شود. این متد همچنین اجازه تسویه/اصلاحیه توسط Third Party را می‌دهد و نتیجه تراکنش از طریق یک callback بازگردانده می‌شود.

نکته: برای تعیین اینکه تسویه/اصلاحیه توسط Third Party انجام شود یا از طریق اپلیکیشن پرداخت Smart Sep، به منوی **تنظیمات** اپلیکیشن پرداخت Smart Sep رفته و در زیرمنوی **پشتیبان**،

گزینه "**تسویه/اصلاحیه توسط Third Party** "را تغییر دهید.

- وضعیت **فعال** گزینه **تسویه/اصلاحیه توسط ThirdParty**:

حالت هایی که شامل ورودی approveByThird در متد doSaleTransaction میشود؛

بسته به نوع کسب و کار و یا نوع ارائه خدمات، Third Party میتواند تصمیم بگیرد که ارسال تسویه/اصلاحیه به صورت اتوماتیک توسط اپلیکیشن پرداخت Smart Sep و یا اینکه توسط خود اپلیکیشن Third Party انجام شود.

- اگر مقدار true داده شود، تراکنش انجام شده منتظر انجام عملیات تسویه/اصلاحیه از سمت Third Party باقی می‌ماند و از شروع تراکنش جدید جلوگیری می‌شود.
- اگر مقدار false داده شود، ارسال تسویه/اصلاحیه به صورت اتوماتیک توسط اپلیکیشن پرداخت Smart Sep انجام می‌شود، در نتیجه نیاز به عملیاتی از سمت Third Party نخواهد بود، و اپلیکیشن پرداخت آماده شروع تراکنش جدید خواهد بود.

- وضعیت **غیرفعال** گزینه **تسویه/اصلاحیه توسط ThirdParty**:

ورودی approveByThird در متد doSaleTransaction **هرچه باشد بی تاثیر است** و ارسال تسویه/اصلاحیه به صورت اتوماتیک توسط اپلیکیشن پرداخت Smart Sep انجام خواهد شد.

نکته: اگر به هر دلیلی تراکنش مالی در اپلیکیشن پرداخت Smart Sep با خطا مواجه شود و مبلغ از دارنده کارت کسر شود، اپلیکیشن پرداخت Smart Sep به صورت اتوماتیک دستور اصلاحیه مبلغ را ارسال خواهد نمود و نیازی به عملیاتی از سمت Third Party نخواهد بود.

پارامترهای ورودی انجام تراکنش

عنوان	نوع داده	کاربرد
amount	String	مبلغی که باید در تراکنش خرید پردازش شود.
reserveNumber	String	شماره رزروی(شماره فاکتور) که با تراکنش مرتبط است.
approveByThird	Boolean	یک بولین که نشان می‌دهد آیا تسویه توسط Third Party لازم است یا خیر.
transactionCallBack	TransactionCallBack	یک رابط callback که نتیجه تراکنش خرید را مدیریت می‌کند.

پارامترهای خروجی انجام تراکنش

پس از انجام تراکنش، نتیجه از طریق TransactionCallBack مدیریت می‌شود.

نحوه انجام تراکنش خرید

```
val transactionCallBack = object : TransactionCallBack {
                    override fun onReceive(transactionData: TransactionData) {
                        Log.i(TAG, "transactionCallBack onReceive: $transactionData")
                        val intent = Intent(this@MainServicesActivity, ResultActivity::class.java)
                        intent.putExtra(TRANSACTION_DATA, transactionData)
                        startActivity(intent)
                    }
                     override fun onUndeterminedStateOfPreviousTxn(transactionData: TransactionData) {
                     //وجود تراکنش تعیین وضعیت نشده
                     //تراکنش مورد را تعیین وضعیت نمایید

                        Log.i(TAG, "transactionCallBack onUndetermined Txn: $transactionData")
                     }

                    override fun onError(errorCode: String, errorMsg: String) {
                        Log.i(TAG, "transactionCallBack onError: $errorCode :  $errorMsg ")
                    }

                    override fun onCanceled() {
                        Log.i(TAG, "transactionCallBack onCanceled: ")
                    }
                }
```

متد onReceive:

این متد نتیجه تراکنش (اطلاعات تراکنش) انجام شده را باز می‌گرداند، تراکنش ممکن است **موفق** و یا **ناموفق** بوده باشد.

براساس فیلد responseCode از transactionData وضعیت موفق و یا ناموفق بودن تراکنش مشخص خواهد شد:

- اگر مقدار responseCode برابر "00" باشد تراکنش موفق است و Third Party بایستی خدمات خود را به مشتری و یا کاربر ارائه دهد.
- اگر مقدار responseCode هر مقداری غیر از "00" باشد، تراکنش ناموفق است و پیام خطا در فیلد responseMessage بازگردانده خواهد شد.

متد onUndeterminedStateOfPreviousTxn:

زمانی فراخوانی میشود که تراکنش انجام شده **موفق** قبلی، **تعیین تکلیف** (تسویه/اصلاحیه) **نشده** باشد، در این هنگام از شروع تراکنش جدید جلوگیری خواهد شد تا زمانی که تراکنش با اطلاعات بازگردانده شده در این متد، تعیین تکلیف (تسویه/اصلاحیه) شود.

- برای انجام عملیات **تسویه** تراکنش، از متد doApprove220() استفاده نمایید.
- برای انجام عملیات **اصلاحیه** تراکنش، از متد doReverse420() استفاده نمایید.

متد onError:

کد خطا و پیام خطاهای اتفاق افتاده را بازمی‌گرداند. این متد خطاهای عمومی مربوط به اپلیکیشن پرداخت و یا خطاهای پیاده سازی SDK توسط Third Party را برمی گرداند و فارغ از پیام ها و یا خطاهای عملیات انجام تراکنش می باشد.

متد onCanceled:

زمانی فراخوانی می‌شود که تراکنش توسط کاربر لغو شده باشد و یا عملیات با Timeout مواجه شود.

درخواست تسویه تراکنش خرید

متد doApprove220 برای پردازش تسویه یک تراکنش خاص بر اساس شماره مرجع بازیابی (RRN) ارائه شده استفاده می‌شود. این متد با سیستم پرداخت تعامل می‌کند تا تراکنش را تأیید کند و نتیجه را از طریق یک callback برمی‌گرداند. این متد زمانی کاربرد دارد که اجازه انجام این تراکنش در اپلیکیشن پرداخت توسط سوپروایزر داده شده باشد.

در برخی کسب و کارها، Third Party بایستی خدمات خود را پس از انجام پرداخت موفق ارائه نماید، اگر موفق به ارائه خدمات شد حتما باید این متد را فراخوانی نماید، در غیراینصورت از شروع تراکنش جدید جلوگیری خواهد شد.

پارامترهای ورودی درخواست تایید تراکنش

عنوان	نوع داده	کاربرد
rrn	String	شماره مرجع بازیابی (RRN) که به طور منحصر به فرد تراکنش مورد نظر برای تأیید را شناسایی می‌کند.
resultCallBack	ResultCallBack	یک رابط callback که نتیجه فرآیند تأیید را مدیریت می‌کند.

پارامترهای خروجی درخواست تایید تراکنش

پس از درخواست تایید تراکنش، نتیجه تراکنش از طریق رابط ResultCallBack اطلاع‌رسانی می‌شود.

نحوه استفاده از درخواست تایید تراکنش

```
val resultCallBack = object : ResultCallBack {
    override fun onSuccess() {
        Log.i(TAG, "CallBack onSuccess ")
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "CallBack onError: $errorCode :  $errorMsg ")
    }
}

sdkManager.doApprove220(rrn, ResultCallBack )
```

درخواست اصلاحیه تراکنش خرید

متد doReverse420 برای پردازش اصلاحیه یک تراکنش خاص با شماره شناسایی، استفاده می‌شود. این متد تضمین می‌کند که تراکنش باطل شده و نتیجه از طریق یک callback اطلاع‌رسانی می‌شود.

اگر تراکنش خرید موفق شده باشد و به هر دلیل اپلیکیشن Third Party موفق به ارائه خدمات خود نشود، اپلیکیشن ThirdParty بایستی تراکنش اصلاحیه را فراخوانی کند و مبلغ به حساب دارنده کارت بازگردد.

پارامترهای ورودی درخواست اصلاحیه تراکنش

عنوان	نوع داده	کاربرد
trace	String	شماره ردیابی که به طور منحصر به فرد تراکنش مورد نظر برای اصلاحیه را شناسایی می‌کند.
resultCallBack	ResultCallBack	یک رابط callback برای مدیریت نتیجه فرآیند اصلاحیه.

پارامترهای خروجی درخواست اصلاحیه تراکنش

پس از درخواست اصلاحیه تراکنش، نتیجه تراکنش از طریق رابط ResultCallBack اطلاع‌رسانی می‌شود.

نحوه استفاده از درخواست اصلاحیه تراکنش

```
val resultCallBack = object : ResultCallBack {
    override fun onSuccess() {
        Log.i(TAG, "CallBack onSuccess ")
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "CallBack onError: $errorCode :  $errorMsg ")
    }
}
sdkManager.doReverse420(traceNumber, ResultCallBack)
```

درخواست تراکنش خدماتی

متد doServiceTransaction برای انجام یک تراکنش خدماتی(خرید شارژ، قبض) بر اساس نوع درخواست خاص استفاده می‌شود.
این متد از تسویه توسط Third Tarty پشتیبانی می‌کند و نتیجه تراکنش از طریق یک callback بازگردانده می‌شود.

پارامترهای ورودی درخواست تراکنش خدماتی

عنوان	نوع داده	کاربرد
requestType	RequestType	نوع درخواست خدماتی که باید پردازش شود. RequestType: REQUEST_TYPE_CHARGE: خرید شارژ REQUEST_TYPE_BILL: پرداخت قبض
approveByThird	Boolean	یک بولین که نشان می‌دهد آیا تسویه توسط third party لازم است یا خیر.
resultCallBack	ResultCallBack	یک رابط callback برای مدیریت نتیجه تراکنش خدماتی.

نکته: در نسخه حال حاضر، در پروژه **Sep**، تسویه تراکنش های خدماتی، توسط اپلیکیشن پرداخت انجام میشود و نیازی به ارسال تسویه توسط Third Party نیست.

پارامترهای خروجی درخواست تراکنش خدماتی

پس از درخواست تراکنش خدماتی، نتیجه تراکنش از طریق رابط ResultCallBack اطلاع‌رسانی می‌شود.

نحوه انجام تراکنش خدماتی

```
val resultCallBack = object : ResultCallBack {
    override fun onSuccess() {
        Log.i(TAG, "CallBack onSuccess ")
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "CallBack onError: $errorCode :  $errorMsg ")
    }
}

sdkManager.doServiceTransaction(RequestType.REQUEST_TYPE_CHARGE, false,
transactionCallBack)
```

استعلام موجودی

متد inquiryBalance برای درخواست موجودی فعلی مرتبط با یک حساب یا کارت استفاده می‌شود. نتیجه تراکنش از طریق یک callback ارائه می‌شود.

پارامترهای ورودی استعلام موجودی

عنوان	نوع داده	کاربرد
transactionCallBack	TransactionCallBack	یک رابط callback برای مدیریت نتیجه درخواست موجودی.

پارامترهای خروجی استعلام موجودی

پس از استعلام موجودی، نتیجه از طریق TransactionCallBack مدیریت می‌شود.

نحوه استفاده از استعلام موجودی

```
val transactionCallBack = object : TransactionCallBack {
    override fun onReceive(transactionData: TransactionData) {
        Log.i(TAG, "transactionCallBack onReceive: $transactionData")
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "transactionCallBack onError: $errorCode :  $errorMsg ")
    }

    override fun onCanceled() {
        Log.i(TAG, "transactionCallBack onCanceled: ")
    }
}
sdkManager.inquiryBalance(transactionCallBack)
```

استعلام اطلاعات تراکنش

متد inquiryTransactionData برای بازیابی داده‌های یک تراکنش خاص بر اساس نوع استعلام و شناسه استفاده می‌شود. نتیجه به صورت انتخابی چاپ و از طریق یک callback بازگردانده می‌شود.

پارامترهای ورودی استعلام اطلاعات تراکنش

عنوان	نوع داده	کاربرد
inquiryType	TxnInquiryType	نوع Enum inquiry که باید انجام شود. (بر اساس rrn، بر اساس trace، بر اساس reserve number)
inquiryId	String	با توجه به inquiry type، id را وارد می کنیم.
printReceipt	Boolean	آیا رسید تراکنش چاپ شود یا خیر.
transactionCallBack	TransactionCallBack	یک رابط callback برای مدیریت نتیجه پرس‌وجوی تراکنش.

پارامترهای خروجی استعلام اطلاعات تراکنش

پس از استعلام، نتیجه تراکنش از طریق transactionCallBack مدیریت می‌شود.

TxnInquiryType به صورت enum تعریف شده است، که براساس نیاز میتوانید مقدار آنرا تغییر دهید:

- REQUEST_TYPE_INQUIRY_BY_RRN: استعلام از طریق شماره مرجع rrn
- REQUEST_TYPE_INQUIRY_BY_TRACE: استعلام از طریق شماره پیگیری trace
- REQUEST_TYPE_INQUIRY_BY_RESERVE_NUMBER: استعلام از طریق شماره فاکتور reserve_number

نحوه استفاده از استعلام اطلاعات تراکنش

```
val transactionCallBack = object : TransactionCallBack {
    override fun onReceive(transactionData: TransactionData) {
        Log.i(TAG, "transactionCallBack onReceive: $transactionData")
        val intent = Intent(this@MainServicesActivity, ResultActivity::class.java)
        intent.putExtra(TRANSACTION_DATA, transactionData)
        startActivity(intent)
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "transactionCallBack onError: $errorCode :  $errorMsg ")
    }

    override fun onCanceled() {
        Log.i(TAG, "transactionCallBack onCanceled: ")
    }
}
//شناسه برای استعلام تراکنش
val trace = "000069"
val rrn = "320138312569"
val reserveNumber = "123465798"

val inquiryType = TxnInquiryType.REQUEST_TYPE_INQUIRY_BY_RRN

sdkManager.inquiryTransactionData(inquiryType, rrn, true, transactionCallBack)
```

درخواست پرینت bitmap (چاپ رسید)

متد printBitmap در Thunder Smart SDK برای چاپ یک تصویر Bitmap به طور مستقیم از یک برنامه اندرویدی استفاده می‌شود. این تابع به توسعه‌دهندگان این امکان را می‌دهد تا یک تصویر را به چاپگر ارسال کنند تا به صورت فیزیکی چاپ شود.

نکات:

1. لطفاً توجه داشته باشید که عرض فایل Bitmap ارسالی حتماً باید 384 پیکسل باشد. در غیر این صورت، اگر عرض فایل بیشتر یا کمتر از این مقدار باشد، ممکن است چاپ دچار دفرمگی شود.
2. لطفاً توجه داشته باشید که محدودیت‌هایی در حجم و اندازه فایل‌های ارسالی وجود دارد. اگر حجم فایل ارسالی زیاد باشد، ممکن است چاپ نشود و با خطا مواجه شوید.

پارامترهای ورودی درخواست پرینت bitmap

عنوان	نوع داده	کاربرد
bitmap	Bitmap	تصویر بیت‌مپ که باید چاپ شود.
resultCallBack	ResultCallBack	یک رابط callback برای مدیریت نتیجه عملیات چاپ.

پارامترهای خروجی درخواست چاپ bitmap

پس از درخواست چاپ bitmap، نتیجه عملیات از طریق رابط ResultCallBack اطلاع‌رسانی می‌شود.

نحوه استفاده از درخواست چاپ bitmap

```
val resultCallBack = object : ResultCallBack {
    override fun onSuccess() {
        Log.i(TAG, "CallBack onSuccess ")
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "CallBack onError: $errorCode :  $errorMsg ")
    }
}
// Attention: width of bitmap must be 384 px
val options = BitmapFactory.Options()
options.inScaled = false
val bitmap = BitmapFactory.decodeResource(context.resources, R.drawable.img_384,
options)

sdkManager.printBitmap(bitmap, resultCallBack)
```

درخواست انجام پیکربندی

متد doConfiguration برای انجام عملیات پیکربندی اپلیکیشن پرداخت Smart Sep استفاده میشود، آماده‌سازی آن برای پردازش تراکنش‌ها استفاده می‌شود. نتیجه این پیکربندی از طریق یک callback بازگردانده می‌شود.

نکته: در نسخه حال حاضر، نیازی به انجام این عملیات نمی باشد، و عملیات پیکربندی در منوی تنظیمات و زیر منوی پشتیبان توسط سوپروایزر انجام خواهد گرفت.

پارامترهای ورودی درخواست انجام پیکربندی

عنوان	نوع داده	کاربرد
resultCallBack	ResultCallBack	یک رابط callback برای مدیریت نتیجه تراکنش پیکربندی.

پارامترهای خروجی درخواست انجام پیکربندی

پس از درخواست انجام پیکربندی، نتیجه تراکنش از طریق رابط ResultCallBack اطلاع‌رسانی می‌شود.

نحوه استفاده از درخواست انجام پیکربندی

```
val resultCallBack = object : ResultCallBack {
    override fun onSuccess() {
        Log.i(TAG, "CallBack onSuccess ")
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "CallBack onError: $errorCode :  $errorMsg ")
    }
}

sdkManager.doConfiguration(resultCallBack)
```

استعلام اطلاعات دستگاه

متد inquiryPosData برای دریافت اطلاعات دستگاه و پذیرنده استفاده می‌شود. اطلاعات دریافت شده از طریق یک callback بازگردانده می‌شود.

پارامترهای ورودی استعلام اطلاعات دستگاه

عنوان	نوع داده	کاربرد
posDataCallBack	PosDataCallBack	یک رابط callback برای مدیریت نتیجه درخواست داده‌های POS.

پارامترهای خروجی استعلام اطلاعات دستگاه

پس از استعلام اطلاعات دستگاه، نتیجه از طریق posDataCallBack مدیریت می‌شود.

نحوه استفاده از استعلام اطلاعات دستگاه

```
val posDataCallBack = object : PosDataCallBack {
    override fun onReceive(posData: PosData) {
        Log.i(TAG, "posDataCallBack onReceive: $posData")
        val intent = Intent(this@MainServicesActivity,ResultActivity::class.java)
        intent.putExtra(POS_DATA, posData)
        startActivity(intent)
    }

    override fun onError(errorCode: String, errorMsg: String) {
        Log.i(TAG, "posDataCallBack onError: $errorCode :  $errorMsg ")
    }
}

sdkManager.inquiryPosData(posDataCallBack)
```