Geidea Gateway Android SDK Integration Guide

**Version 3.0.0**

|  |  |
| --- | --- |
| Document version | 3.0.0 |
| Date of version | 5 Sep 2022 |
| Author | Svetlozar Kostadinov |

|  |
| --- |
|  |

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes (More details in CHANGELOG.MD) |
| 1.0 | 24.03.2021 | Svetlozar Kostadinov | Initial version |
| 2.0 | 06.01.2022 | Svetlozar Kostadinov | 3DS v2, Payment Intents, e-Invoices, Meeza QR, Arabic localization |
| 3.0.0 | 05.09.2022 | Svetlozar Kostadinov | 3DS token payments Buy Now Pay Later (BNPL) payments: - ValU Installments - Shahry Installments - Souhoola Installments |

Contents

[Introduction 4](#_Toc104846099)

[1.1 Document Purpose 4](#_Toc104846100)

[Android SDK Integration 5](#_Toc104846101)

[1.2 Requirements 5](#_Toc104846102)

[1.3 How to start 5](#_Toc104846103)

[1.3.1 Gradle setup 5](#_Toc104846104)

[1.3.2 SDK Initialization 5](#_Toc104846105)

[1.4 Getting your Merchant configuration 6](#_Toc104846106)

[1.5 Integration types overview 6](#_Toc104846107)

[1.6 The “UI Flow” concept 6](#_Toc104846108)

[1.7 Payment flow 7](#_Toc104846109)

[1.7.1 Building your PaymentIntent 7](#_Toc104846110)

[1.7.2 Starting Payment flow 8](#_Toc104846111)

[1.7.3 Receiving the Order result 9](#_Toc104846112)

[1.7.4 PaymentIntent properties specification 9](#_Toc104846113)

[1.7.5 PaymentMethod properties specification 12](#_Toc104846114)

[1.8 API 13](#_Toc104846115)

[Note: full developer reference of the API is the provided KDoc / Javadoc packages. 13](#_Toc104846116)

[1.8.1 GeideaPaymentSdk 13](#_Toc104846117)

[1.8.2 GatewayApi 13](#_Toc104846118)

[1.8.3 PaymentIntentApi 13](#_Toc104846119)

[1.8.4 Buy Now Pay Later (BNPL) API 14](#_Toc104846120)

[1.9 UI Components 15](#_Toc104846121)

[1.9.1 Introduction 15](#_Toc104846122)

[1.9.2 Types 15](#_Toc104846123)

[1.9.3 Validation of forms 15](#_Toc104846124)

[1.9.4 Validation of input fields 16](#_Toc104846125)

[1.9.5 Styling 17](#_Toc104846126)

[1.9.6 Simple payment form with PaymentFormView 18](#_Toc104846127)

[1.9.7 Custom payment form by composing Geidea components 22](#_Toc104846128)

[1.9.7.1 CardInputView 22](#_Toc104846129)

[1.9.7.2 AddressInputView 24](#_Toc104846130)

[1.9.8 Custom payment form with non-Geidea components 26](#_Toc104846131)

[1.9.9 Custom integration 26](#_Toc104846132)

[1.10 Code samples 27](#_Toc104846133)

[1.11 Response codes 28](#_Toc104846134)

[1.11.1 Response codes and messages 28](#_Toc104846135)

[1.11.2 Detailed response code and message groups: 28](#_Toc104846136)

[Appendix A. Glossary 34](#_Toc104846137)

# Introduction

## Document Purpose

The purpose of this Integration guide is to serve as a technical documentation for merchants who wish to integrate Geidea Payment SDK for Android so that they can use Payment Gateway services in their Android application.

This guide describes the functionality and APIs provided by the SDK and different approaches to integrate the SDK and customize it.

# Android SDK Integration

## Requirements

* Android 6.0+
* Kotlin or Java 8

## How to start

### Gradle setup

1. Add GitHub Packages as a Maven repository to your project-level build.gradle

allprojects {  
 maven {  
 url = uri("https://maven.pkg.github.com/GeideaSolutions/Android-SDK")  
 credentials {  
 username = project.findProperty("gpr.user")  
 password = project.findProperty("gpr.key")  
 }  
 }  
}

2. Define your GitHub username (gpr.user) and GitHub Personal Access Token (gpr.key) in your gradle.properties file (without the pointy brackets):

gpr.user=<YOUR GITHUB USSERNAME>  
gpr.key=<YOUR GITHUB PERSONAL ACCCESS TOKEN>

You can see your Personal Access Token (PAT) in your  
GitHub Profile > Settings > Developer Options > Personal access tokens.  
If you do not have one yet, you can generate it as explained [here](https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token).  
  
3. Add the SDK as a dependency in your app-level build.gradle file

implementation 'net.geidea.paymentsdk:paymentsdk:<LATEST VERSION>'

Now Geidea Payment SDK should be imported in your project.

### SDK Initialization

As an initialization step the SDK expects that you provide your Merchant credentials with the GeideaPaymentSdk.setCredentials() method. However, it is not required to set them on each start. It could be once per installation of the app as the credentials are persisted securely encrypted on device. You can check if there are credentials already stored with the GeideaPaymentSdk.hasCredentials property. It is important that they stored before using the SDK.

**Example**

if (!GeideaPaymentSdk.hasCredentials) {

GeideaPaymentSdk.setCredentials(

merchantKey = "<YOUR MERCHANT KEY>",

merchantPassword = "<YOUR MERCHANT PASSWORD>"

)

}

**IMPORTANT**  
Do not hard-code your merchant password directly into your APK. Instead get it dynamically (from an endpoint of your backend or elsewhere) due to security reasons.

## Getting your Merchant configuration

After the SDK initialization and before performing transactions it might be useful to obtain your current Merchant configuration.

**Kotlin**

Use GatewayApi.getMerchantConfiguration() suspend function. Note that you do not need to call it inside IO context because this is done automatically by the SDK.

coroutineScope.launch(context = Dispatchers.Main) **{** // Automatically called on Dispatchers.IO context  
 val configuration = GatewayApi.getMerchantConfiguration()  
**}**

**Java**

GatewayApi.getMerchantConfiguration(result -> {  
 // Handle success/error result  
});

## Integration types overview

There are few different approaches to integrate Geidea payment services into your app. The choice what approach to follow depends mainly on your needs for UI customizability.

1. **Simple** integration – SDK hosts the entire UI flow and performs all transactions. A “turnkey” solution that requires minimal setup. You simply call a method to start the Payment flow and then receive your Order after everything is ready. For more info refer to [“UI Flow” concept](https://endava-my.sharepoint.com/personal/svetlozar_kostadinov_endava_com/Documents/Geidea/Releases/deliverable-Jan6-2022/UI_Flow#_) and [Payment flow](#_Payment_flow).
2. **Mixed** integration – your Merchant app hosts the payment form UI. SDK performs the authentication and all transactions.
3. **Custom** integration – the Merchant app hosts entire UI flow (payment form, authentication) and performs all transactions by calling the Direct [API](#_API).

## The “UI Flow” concept

SDK uses the concept of UI flow which is a sequence of UI screens, network calls and various other operations all encapsulated in one Activity call for result. UI flows are implemented based on the typical Android Activity results where one SDK activity (or a chain of more than one) is launched with an **input** intent, then it performs its work and finally produces some **output** which contains the result data you are interested in (usually the Order). A flow is represented and managed by an ActivityContract implementation.

**Using Activity result contracts**

Instead of relying on the traditional and now deprecated startActivityForResult() method the SDK embraces the newer Activity Result APIs which offer some benefits for you. For more info please visit <https://developer.android.com/training/basics/intents/result>.

## Payment flow

The Payment flow expects an input of type PaymentIntent and returns a result of type GeideaResult<Order> at the end of the payment process. The PaymentContract is used to manage the input/output parcelization.

Declare a launcher somewhere in your code from where you wish to start the payment…

**private var paymentLauncher**: ActivityResultLauncher<PaymentIntent>

…and then register it with a PaymentContract instance and your function or lambda that should accept the final result.

**fun** handleOrderResult(result: GeideaResult<Order>) {

*/\*\* Handle the order response here \*/*

}

**paymentLauncher** = registerForActivityResult(PaymentContract(), ::handleOrderResult)

### Building your PaymentIntent

PaymentIntent contains details about the order, customer and preferred payment method. It has two mandatory properties – amount and currency.

**Kotlin**

**val paymentIntent** = *PaymentIntent* **{** *// Mandatory properties* **amount** = 123.45  
 **currency** = **"SAR"**

*// Optional properties* **paymentMethod** = PaymentMethod **{** cardHolderName = **"John Doe"** cardNumber = **"5123450000000008"** expiryDate = ExpiryDate(month = 1, year = 25)  
 cvv = **"123"  
 }****callbackUrl** = **"https://website.hook/"  
 merchantReferenceId** = **"1234"  
 customerEmail** = **"email@test.com"  
 billingAddress** = Address(  
 countryCode = **"SAU"**,  
 city = **"Riyadh"**,  
 street = **"Street 1"**,  
 postCode = **"1000"** )  
 **shippingAddress** = Address(  
 countryCode = **"SAU"**,  
 city = **"Riyadh"**,  
 street = **"Street 1"**,  
 postCode = **"1000"** )  
**}**

**Java**

PaymentIntent paymentIntent = **new** PaymentIntent.Builder()  
 .setAmount(123.45d)  
 .setCurrency(**"SAR"**)  
 .setPaymentMethod(**new** PaymentMethod.Builder()  
 .setCardHolderName(**"John Doe"**)  
 .setCardNumber(**"5123450000000008"**)  
 .setExpiryDate(**new** ExpiryDate(1, 25))  
 .setCvv(**"123"**)  
 .build()  
 )  
 .setCallbackUrl(**"https://website.hook/"**)  
 .setMerchantReferenceId(**"1234"**)  
 .setCustomerEmail(**"email@test.com"**)  
 .setBillingAddress(**new** Address(  
 **"SAU"**,  
 **"Riyadh"**,  
 **"Street 1",**  
 **"1000"** ))  
 .setShippingAddress(**new** Address(  
 **"SAU"**,  
 **"Riyadh"**,

**"Street 1",**  
 **"1000"** ))  
 .build();

**Card data input**

The Payment flow supports two ways of collecting the card data:

1. **By the SDK** – when the paymentMethod property is not set or set to null. In this case the SDK will display a hosted native payment form to collect the card and address data.
2. **By the Merchant** – when you supply paymentMethod value. In this case you are responsible to collect the card data through your app UI and hand it to the SDK. The SDK will not show a payment form but will directly perform the necessary transactions and authentications.

**Validations**

Multiple basic validation checks are performed on construction of PaymentIntent and PaymentMethod. E.g. check if the CVV is 3 or 4 digits. If some validation is not satisfied an IllegalArgumentException with a message is thrown. For a comprehensive list of validity conditions please refer to Section 3.6. The full validation is performed server-side and FieldValidationError is returned on bad input.

### Starting Payment flow

Example:

paymentLauncher.launch(paymentIntent)

### Receiving the Order result

The final result of the Payment flow is returned as a sealed object of type GeideaResult<Order>.

**Example**

**fun** handleOrderResult(result: GeideaResult<Order>) {  
 **when** (result) {  
 **is** GeideaResult.Success<Order> -> {  
 *// Payment successful, order returned in result.data* }  
 **is** GeideaResult.Error -> {  
 **when** (result) {   
 **is** GeideaResult.NetworkError -> {  
 *// Client or server error* handleNetworkError(  
 result.responseCode  
 result.responseMessage  
 result.detailedResponseCode  
 result.detailedResponseMessage  
 )  
 }  
 **is** GeideaResult.SdkError -> {  
 *// An unexpected error due to improper SDK  
 // integration or SDK internal bug* handleSdkError(result.errorCode, result.errorMessage)  
 }  
 }  
 }  
 **is** GeideaResult.Cancelled -> {  
 *// Payment flow cancelled by the user (e.g. Back button)*   
 }  
 }  
}

### PaymentIntent properties specification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter name | Description | Type | Validations | Optionality |
| amount | Transaction amount | BigDecimal | Must be decimal, positive and must not more than 2 digits after decimal point | Mandatory |
| currency | Currency of the amount, standard:  ISO 4712 currency code | String | Must be 3 characters long | Mandatory |
| paymentMethod | Plain card data. Supply this if you collect the card data in your app and use the SDK only to perform authentication and transactions | PaymentMethod | paymentMethod is mutually exclusive with the tokenId property | Optional |
| tokenId | The id of a previously tokenized card to pay with.  See cardOnFile property |  | tokenId is mutually exclusive with the paymentMethod property.  Depending on MerchantConfigurationResponse. isCvvRequiredForTokenPayments and MerchantConfigurationResponse. is3dsRequiredForTokenPayments the CVV input and authentication screens might be skipped.  **Note**: For Egypt the authentication for token payments is always required. | Optional |
| paymentMethods | List of payment method names that you wish to limit this concrete payment to. If not set, all methods in your merchant configuration will be available. If list has a single payment method, then the Payment method selection screen is skipped, and the UI flow starts with this payment method | List<String> | Payment method names:   * visa * mastercard * amex * mada * meeza * qrcode (Meeza QR, Egypt only) * valu (BNPL, Egypt only) * shahry (BNPL, Egypt only) * souhoola (BNPL, Egypt only)   See CardBrand. allSupportedBrands | Optional |
| paymentOperation | If null the effective value is read from merchant configuration | String | Possible values:   * null, * “Pay” * “PreAuthorize” * “AuthorizeCapture” | Optional |
| initiatedBy |  |  | Possible values:   * “Internet”, * “Merchant” | ? |
| callbackURL | The response with order details, will be returned to this URL | String | Must be a valid URL and to have an HTTPS protocol | Optional |
| merchantReferenceId | Use this as your unique reference for each transaction | String | \* | Optional |
| customerEmail | Customer email if you already have it | String | Must be a valid email address | Optional |
| showCustomerEmail | Show the customer e-mail in the card payment form and allow the customer to override it | Boolean | \* | Optional |
| billingAddress  For billing address – the following parameters can be passed:   * countryCode * street * city * postCode | Billing address details for thecustomer if you already have it  country standard:  ISO 3166 – alpha-3 code | String | **countryCode -** must be 3 characters  **street, city, postcode –** max 255 characters | Optional |
| shippingAddress  For shipping address – the same parameters as for billing can be passed:   * countryCode * street * city * postCode | Shipping address details for thecustomer if you already have it  country standard:  ISO 3166 – alpha-3 code | String | **countryCode -** must be 3 characters  **street, city, postcode –** max 255 characters | Optional |
| showBillingAddress | Show the customer billing address in the card payment form and allow the customer to override it | Boolean | \* | Optional |
| showShippingAddress | Show the customer shipping address in the card payment form and allow the customer to override it | Boolean | \* | Optional |
| showReceipt | Shows a transaction receipt screen with transaction details after success or show error reason in case of failure | Boolean | If not set, the value of isTransactionReceiptEnabled  flag of your merchant configuration will be used | Optional |
| cardOnFile | Allow/disallow tokenization of the payment card (optional). Default is false. | Boolean | Has effect only if the tokenization is enabled in your Merchant configuration. | Optional |
| agreementType | Type of agreement between Merchant and customer on future use of a tokenized card | String | Possible values:   * null, * “Recurring”, * “Installment”, * “Unscheduled” | Mandatory if agreementId is not null |
| agreementId | Agreement ID | String |  | Mandatory if agreementType is not null |
| eInvoiceId | e-Invoice ID | String | e-Invoice can be created with PaymentIntentApi.createEInvoice() | Optional |
| orderItems | Cart items for this order | List<OrderItem> | The total sum of the items prices multiplied by their quantities must be equal to PaymentData.amount | Mandatory for *Shahry* and *Souhoola* payment methods |

\*Please note that all above String parameters have a maximal length of 255 symbols.

### PaymentMethod properties specification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter name | Description | Type | Client-side Validations | Optionality |
| cardHolderName | The name appearing on the physical card | String | 5..255 length | Mandatory |
| cardNumber | Card PAN | String | Non-empty | Mandatory |
| expiryDate | Card expiry date | ExpiryDate | - | Mandatory |
| expiryDate.month | Card expiry month | Int | 1..12 | Mandatory |
| expiryDate.year | Card expiry year | Int | 1..99 | Mandatory |
| owner |  | String | - | Optional |

## API

**Note**: full developer reference of the API is the provided KDoc / Javadoc packages.

### GeideaPaymentSdk

GeideaPaymentSdk is the main SDK class. It provides methods to initialize the SDK. See [SDK Initialization](#_SDK_Initialization).

**Managing your Merchant credentials**

fun setCredentials(merchantKey: String, merchantPassword: String)  
fun hasCredentials: Boolean  
fun clearCredentials()

**Cleaning up**

fun cleanup()

### REST client with Kotlin and Java API

Geidea Payment SDK exposes 2 different types of public API methods for accessing the server REST endpoints. These methods are located in packages under net.geidea.paymentsdk.api.\*:

1. As **Kotlin suspend functions** working by default on the IO Dispatcher:
   * net.geidea.paymentsdk.api.gateway.GatewayApi
   * net.geidea.paymentsdk.api.paymentintent.PaymentIntentApi
   * net.geidea.paymentsdk.api.installments.cnp.valu.ValuInstallmentsApi
   * net.geidea.paymentsdk.api.installments.cnp.shahry.ShahryInstallmentsApi
   * net.geidea.paymentsdk.api.installments.cnp.souhoola.SouhoolaInstallmentsApi
2. As static methods accepting **Java callback** as an argument. The network call is performed on an IO thread and the result is passed to the callback on the Main thread:
   * net.geidea.paymentsdk.api.gateway.GatewayCallbackApi
   * net.geidea.paymentsdk.api.paymentintent.PaymentIntentCallbackApi
   * net.geidea.paymentsdk.api.installments.cnp.valu.ValuInstallmentsCallbackApi
   * net.geidea.paymentsdk.api.installments.cnp.shahry.ShahryInstallmentsCallbackApi
   * net.geidea.paymentsdk.api.installments.cnp.souhoola.SouhoolaInstallmentsCallbackApi

### GatewayApi

**Getting your merchant configuration**

suspend fun getMerchantConfiguration(): GeideaResult<MerchantConfigurationResponse>

**Performing transactions**

suspend fun authenticate(authenticationRequest: AuthenticationRequest): GeideaResult<AuthenticationResponse>

suspend fun pay(paymentRequest: PaymentRequest): GeideaResult<Order>

suspend fun payWithToken(tokenPaymentRequest: TokenPaymentRequest): GeideaResult<Order>

suspend fun capture(captureRequest: CaptureRequest): GeideaResult<Order>

**Order API**

suspend fun getOrders(orderSearchRequest: OrderSearchRequest): GeideaResult<OrderSearchResponse>

suspend fun getOrder(orderId: String): GeideaResult<Order>

suspend fun captureOrder(captureRequest: CaptureRequest): GeideaResult<Order>  
suspend fun refundOrder(refundRequest: RefundRequest): GeideaResult<Order>  
suspend fun cancelOrder(cancelRequest: CancelRequest): GeideaResult<PaymentResponse>

### PaymentIntentApi

**e-Invoice API**

suspend fun createEInvoice(createEInvoiceRequest: CreateEInvoiceRequest):GeideaResult<EInvoiceResponse>

suspend fun getEInvoice(eInvoiceId: String): GeideaResult<EInvoiceResponse>

suspend fun updateEInvoice(updateEInvoiceRequest: UpdateEInvoiceRequest):GeideaResult<EInvoiceResponse>  
suspend fun deletePaymentIntent(paymentIntentId: String): GeideaResult<EInvoiceResponse>

**Meeza QR API**

suspend fun createMeezaPaymentQrCode(createPaymentIntentRequest: CreateMeezaPaymentIntentRequest): GeideaResult<MeezaQrImageResponse>  
suspend fun sendMeezaRequestToPay(meezaPaymentRequest: MeezaPaymentRequest): GeideaResult<MeezaPaymentResponse>

**Order Receipt API**

suspend fun getOrderReceipt(orderId: String): GeideaResult<ReceiptResponse>

### BNPL API

BNPL (Buy Now Pay Later) APIs are REST accessor methods that could be used to create **custom** BNPL payment UI/UX flows. Note that called these BNPL API methods is required only if you need fully custom UI/UX flow. In the majority of use-cases it is recommended to use the [**simple** integration](#_Integration_types_overview) which exposes a complete payment UI ready to use out-of-the-box.

**What is BNPL?**

BNPL payment methods are deferred payments with which the customer can order something immediately but pay later in the form of monthly **installments**. But you as a merchant receive the total amount immediately – not later. It is up to the customer to select the installment plan which best suits them.

**Tenure / tenor months**

The number of monthly installments is called *tenure* or *tenor months*. Different BNPL providers provide different range of possible tenures normally varying from 6 months to 5 years.

**Down payment**

The customer is given the choice whether to pay part of the total amount upfront. This amount is called *down payment* amount. The difference between the *order amount* and the *down payment* amount is called *financed amount* and is paid immediately to you by the concrete BNPL provider while the customer assumes a debt to the BNPL provider. No matter what kind of down payment the customer chooses, you as a merchant receive the total amount immediately after the BNPL order is successful. Depending on the installment plan selected by the customer there might be or not a *purchase fee* (a.k.a. *administrative fee*) which could be paid upfront together with the *down payment* (if any). *Down payment* must be performed as a card (auth + pay) transaction or Meeza QR transaction in the BNPL order. Once the down payment transaction is successfully executed, the BNPL order flow can proceed (normally with OTP confirmation).

**One-Time Password (OTP)**

OTP confirmation is used as the last step in each BNPL flow. It is a short code normally sent via SMS or in the app of the BNPL provider and the customer has a brief time window to enter it.

**Flows**

Each BNPL provider has a sequence (flow) of SDK calls that are necessary to for an order to be processed successfully. Some of the properties returned in the responses are used in the next calls.

#### ValuInstallmentsApi

Before using ValuApi the isValuBnplEnabled configuration flag must be checked. The minimal amount supported by *ValU Installments* can be obtained from the valUMinimumAmount property of the merchant configuration.

**Methods**

suspend fun verifyCustomer(request: VerifyCustomerRequest): GeideaResult<VerifyCustomerResponse>

suspend fun getInstallmentPlans(request: InstallmentPlansRequest): GeideaResult<InstallmentPlansResponse>

suspend fun selectInstallmentPlan(request: SelectInstallmentPlanRequest): GeideaResult<SelectInstallmentPlanResponse>

suspend fun generateOtp(request: GenerateOtpRequest): GeideaResult<GenerateOtpResponse>

suspend fun purchase(request: ConfirmRequest): GeideaResult<ConfirmResponse>

**Kotlin example**

**Step 1.** Verify customer by phone number

val verifyResult = ValuInstallmentsApi.verifyCustomer(  
 VerifyCustomerRequest(customerIdentifier = **"0123456789"**) // Customer input - phone number  
)

**Step 2.** Get a list of available installment plans

val plansResult = ValuInstallmentsApi.getInstallmentPlans(  
 *InstallmentPlansRequest* **{** customerIdentifier = **"0123456789"** // Customer input – phone number  
 totalAmount = BigDecimal(1200)  
 currency = **"EGP"** downPayment = BigDecimal.*ZERO* // Customer input  
 giftCardAmount = BigDecimal.*ZERO* // Customer input  
 campaignAmount = BigDecimal.*ZERO* // Customer input  
 **}**)

**Step 3.** Customer selects an installment plan from the list

val selectedPlanIndex = ... // Index of the selected plan in the UI  
val selectedPlan: InstallmentPlan = plansResult.data.installmentPlans[selectedPlanIndex]  
  
val selectedPlanResult = ValuInstallmentsApi.selectInstallmentPlan(  
 *SelectInstallmentPlanRequest* **{** totalAmount = BigDecimal(1200)  
 currency = **"EGP"** tenure = selectedPlan.tenorMonth  
 adminFees = selectedPlan.adminFees // 36 EGP in this example  
 giftCardAmount = BigDecimal.*ZERO* // Customer input  
 campaignAmount = BigDecimal.*ZERO* // Customer input  
 **}**)

**Step 4.** Process down payment (this step must be skipped if down payment + fees == 0)

val paymentLauncher = activity.registerForActivityResult(PaymentContract(), ::onPaymentResult)  
paymentLauncher.launch(  
 *PaymentData* **{** orderId = selectedPlanResult.data.orderId  
 amount = BigDecimal(100 + 36) // Down payment + purchase fees  
 currency = **"EGP"** paymentMethods = *setOf*(**"visa"**, **"mastercard"**, **"amex"**, **"mada"**, **"meeza"**, **"qrcode"**)  
 **}**)  
  
// Handle the down payment result  
fun onPaymentResult(downPaymentResult: GeideaResult<Order>) {  
 if (downPaymentResult is GeideaResult.Success) {  
 // Proceed to next step of BNPL flow (step 5)  
 }  
}

**Step 5.** Generate and send the OTP

val otpResult = ValuInstallmentsApi.generateOtp(  
 GenerateOtpRequest(  
 customerIdentifier = **"0123456789"**,  
 bnplOrderId = plansResult.data.bnplOrderId  
 )  
)

**Step 6.** Confirm the OTP

val confirmResult = ValuInstallmentsApi.confirm(  
 ConfirmRequest **{** customerIdentifier = **"0123456789"** orderId = selectedPlanResult.data.orderId  
 bnplOrderId = plansResult.data.bnplOrderId  
 otp = **"123456"**, // Customer input: 6 digits  
 totalAmount = BigDecimal(1200)  
 currency = **"EGP"** adminFees = BigDecimal(36)  
 downPayment = BigDecimal(100)  
 giftCardAmount = BigDecimal.ZERO  
 campaignAmount = BigDecimal.ZERO  
 tenure = selectedPlan.tenorMonth  
 **}**)

#### ShahryInstallmentsApi

Before using **ShahryInstallmentsApi** the isShahryCnpBnplEnabled configuration flag must be checked.

**Methods**

suspend fun selectInstallmentPlan(request: SelectInstallmentPlanRequest): GeideaResult<SelectInstallmentPlanResponse>

suspend fun confirm(request: ConfirmRequest): GeideaResult<ConfirmResponse>

**Kotlin example**

**Step 1.** Select an installment plan

val selectedPlanResult = ShahryInstallmentsApi.selectInstallmentPlan(  
 *SelectInstallmentPlanRequest* **{** customerIdentifier = **"0123456789"** // Customer input - phone number  
 totalAmount = BigDecimal(1200)  
 currency = **"EGP"** items = *listOf*(*ShahryOrderItem* **{** merchantItemId = **"id123"** name = **"Phone"** description = **"Phone description"** categories = *listOf*(**"Electronics"**, **"Smartphones"**)  
 count = 2  
 price = BigDecimal(600)  
 currency = **"EGP"  
 }**)  
 **}**)

**Step 2.** Process down payment (this step must be skipped if down payment + fees = 0)

val paymentLauncher = activity.registerForActivityResult(PaymentContract(), ::onPaymentResult)  
paymentLauncher.launch(  
 *PaymentData* **{** orderId = selectedPlanResult.data.orderId  
 amount = BigDecimal(100 + 36) // Down payment + purchase fees  
 currency = **"EGP"** paymentMethods = *setOf*(**"visa"**, **"mastercard"**, **"amex"**, **"mada"**, **"meeza"**, **"qrcode"**)  
 **}**)  
  
// Handle the down payment result  
fun onPaymentResult(downPaymentResult: GeideaResult<Order>) {  
 if (downPaymentResult is GeideaResult.Success) {  
 // Proceed to next step of BNPL flow (step 3)  
 }  
}

**Step 3.** Confirm the purchase

val confirmResult = ShahryApi.confirm(  
 ConfirmRequest(  
 orderId = selectedPlanResult.data.orderId,  
 orderToken = **"<customer token>"** // Customer input, customer can see their token in Shahry app  
 )  
)

#### SouhoolaInstallmentsApi

Before using SouhoolaApi the isSouhoolaCnpBnplEnabled configuration flag must be checked. The minimal amount supported by *Souhoola Installments* can be obtained from the souhoolaMinimumAmount property of the merchant configuration.

**Methods**

suspend fun verifyCustomer(request: VerifyCustomerRequest): GeideaResult<VerifyCustomerResponse>  
suspend fun getInstallmentPlans(request: InstallmentPlansRequest): GeideaResult<InstallmentPlansResponse>  
suspend fun selectInstallmentPlan(request: SelectInstallmentPlanRequest): GeideaResult<SelectInstallmentPlanResponse>  
suspend fun reviewTransaction(request: ReviewTransactionRequest): GeideaResult<ReviewTransactionResponse>  
suspend fun generateOtp(request: GenerateOtpRequest): GeideaResult<GenerateOtpResponse>  
suspend fun confirm(request: ConfirmRequest): GeideaResult<ConfirmResponse>

**Kotlin Example**

**Step 1.** Verify customer by phone number

val verifyResult = SouhoolaInstallmentsApi.verifyCustomer(  
 VerifyCustomerRequest(  
 customerIdentifier = **"0123456789"**, // Customer input - phone number  
 customerPin = **"12345"** // Customer input - 5 digits  
 )  
)

**Step 2.** Get a list of available installment plans

val plansResult = SouhoolaInstallmentsApi.getInstallmentPlans(  
 *InstallmentPlansRequest* **{** customerIdentifier = **"0123456789"** // from step 1  
 customerPin = **"12345"** // from step 1  
 totalAmount = BigDecimal(1200)  
 currency = **"EGP"** downPayment = BigDecimal.*ZERO* // Customer input  
 **}**)

**Step 4.** Customer selects an installment plan from the list

val selectedPlanIndex = ... // Index of the selected plan in the UI  
val selectedPlan: InstallmentPlan = plansResult.data.installmentPlans[selectedPlanIndex]  
  
// Step 5. Customer reviews the payment with the selected installment plan  
  
val reviewResult = SouhoolaInstallmentsApi.reviewTransaction(  
 *ReviewTransactionRequest* **{** customerIdentifier = **"0123456789"** // from step 1  
 customerPin = **"12345"** // from step 1  
 totalAmount = BigDecimal(1200)  
 currency = **"EGP"** downPayment = BigDecimal(100) // from step 2  
 tenure = selectedPlan.tenorMonth  
 minimumDownPaymentTenure = selectedPlan.minDownPayment  
 promoCode = selectedPlan.promoCode  
 approvedLimit = verifyResult.data.approvedLimit // from step 1  
 outstanding = verifyResult.data.outstanding // from step 1  
 availableLimit = verifyResult.data.availableLimit // from step 1  
 minLoanAmount = verifyResult.data.minLoadAmount // from step 1  
 items = *listOf*(  
 *SouhoolaOrderItem* **{** merchantItemId = **"id123"** name = **"Phone"** description = **"Phone description"** categories = **"Electronics, Smartphones"** count = 2  
 price = BigDecimal(600)  
 currency = **"EGP"  
 }** )  
 **}**)

**Step 6.** Confirm the selection of installment plan with a server call

val selectedPlanResult = SouhoolaInstallmentsApi.selectInstallmentPlan(  
 SelectInstallmentPlanRequest **{** customerIdentifier = **"0123456789"** // from step 1  
 customerPin = **"12345"** // from step 1  
 totalAmount = BigDecimal(1200)  
 currency = **"EGP"** bnplDetails = *SouhoolaBnplDetails* **{** // from step 5  
 souhoolaTransactionId = reviewResult.data.souhoolaTransactionId  
 totalInvoicePrice = reviewResult.data.totalInvoicePrice  
 downPayment = reviewResult.data.downPayment  
 loanAmount = reviewResult.data.loanAmount  
 netAdminFees = reviewResult.data.netAdministrativeFees  
 mainAdminFees = reviewResult.data.mainAdministrativeFees  
 annualRate = reviewResult.data.annualRate  
 **}  
 }**)

**Step 7.** Process down payment (this step must be skipped if down payment + fees == 0)

val paymentLauncher = activity.registerForActivityResult(PaymentContract(), ::onPaymentResult)  
paymentLauncher.launch(  
 *PaymentData* **{** orderId = selectedPlanResult.data.orderId // from step 6  
 amount = BigDecimal(100 + 36) // Down payment + purchase fees  
 currency = **"EGP"** paymentMethods = *setOf*(**"visa"**, **"mastercard"**, **"amex"**, **"mada"**, **"meeza"**, **"qrcode"**)  
 **}**)  
  
// Handle the down payment result  
fun onPaymentResult(downPaymentResult: GeideaResult<Order>) {  
 if (downPaymentResult is GeideaResult.Success) {  
 // Proceed to next step of BNPL flow (step 8)  
 }  
}

**Step 8.** Generate and send OTP

val otpResult = SouhoolaInstallmentsApi.generateOtp(  
 *GenerateOtpRequest* **{** customerIdentifier = **"0123456789"** // from step 1  
 customerPin = **"12345"** // from step 1  
 souhoolaTransactionId = reviewResult.data.souhoolaTransactionId // from step 5  
 orderId = selectedPlanResult.data.orderId // from step 6  
 **}**)

**Step 9.** Confirm the OTP

val confirmResult = SouhoolaInstallmentsApi.confirm(  
 *ConfirmRequest* **{** customerIdentifier = **"0123456789"** // from step 1  
 customerPin = **"12345"** // from step 1  
 souhoolaTransactionId = reviewResult.data.souhoolaTransactionId // from step 5  
 orderId = selectedPlanResult.data.orderId // from step 6  
 otp = **"12345"** // Customer input  
 **}**)

## UI

### Introduction

Developing your own payment UI is **not** necessary in case of **Simple** integration type. Everything necessary to perform a payment is integrated in the Payment flow and is working out-of-the-box. However, in the **Mixed** integration type you may embed some or the entire UI inside your app. In the **Custom** integration type your app hosts all the UI and performs the necessary transactions. The SDK provides Material components that make creating of your own payment UI for Mixed and Custom integrations easier compared to creating your own UI from scratch. Many daunting or annoying tasks like card data validation are handled for you internally in the components.

### Components

The SDK includes a set of Material components to help you build a modern payment UI. There are two types of components:

* Form views (composite views)
  + PaymentFormView
  + CardInputView
  + AddressInputView
* Input fields – sub-classes of TextInputEditText
  + CardNumberEditText
  + CardHolderEditText
  + CardExpiryEditText
  + CardSecurityCodeEditText
  + EmailEditText
  + CountryAutoCompleteTextView
  + StreetEditText
  + CityEditText
  + PostCodeEditText

### Validation of forms

Validation is handled by the Form views internally. All Form views expose the Boolean val property isValid which is automatically updated in sync with the input. Listening for validation status changes can be done with the following calls:

paymentFormView.setOnValidationChangedListener { value: PaymentFormData?, valid: Boolean ->

}

cardInputView.setCardInputListener(object : CardInputAdapter() {  
 override fun onCardValidationChanged(cardValid: Boolean) {

*// Called when input data becomes valid or invalid*  
 }  
})

addressInputView.setAddressInputListener(object : AddressInputAdapter() {  
 override fun onAddressValidationChanged(address: Address, valid: Boolean) {

*// Called when input data becomes valid or invalid*  
 }  
})

### Validation of input fields

Geidea Material components delegate the validation of their input to an instance of the Validator interface. At construction the components do not have an attached validator, so assigning an appropriate one with the setValidator() method is responsibility of the higher level component or activity which owns that component.

**Validators**

Validator is a very simple generic interface having a validate() whose only purpose is to check if the current value in the input field is valid or not. If not it returns information for the exact reason why it is invalid. Validators are called on each input change.

**Validation status**

The UI components expose the validation status of their current input as the property

val validationStatus: ValidationStatus

When you are not interested in detailed validation status there is a simpler alternative

val isValid: Boolean

The validationStatus property is updated automatically by the component with the status returned by the associated validator. ValidationStatus is a sealed class with 3 possible sub-types:

* Valid - signifies that the input is acceptable for further processing.

object Valid : ValidationStatus()

* Invalid – signifies that the input is an unacceptable for further processing due to some reason which is specified in the reason property.  
  data class Invalid(val reason: InvalidationReason) : ValidationStatus()
* Undefined – special type for internal use which only means that the validation is not yet performed. Validators must never return this type.

object Undefined : ValidationStatus()

**Listening for (in)validation events**

Use the following two methods to register for validation status changes:

textField.setOnValidStatusListener { value: T ->

textField.updateErrorMessage()

}

setOnInvalidStatusListener { value: T?, invalidStatus: ValidationsStatus.Invalid->

textField.updateErrorMessage()

}

In the example above we call updateErrorMessage() for the field to update its error message from the validation status:

* When valid - the error message is cleared.
* When invalid - the error message is set to the one from reason property.

**Invalidation reasons**

The InvalidationReason subtypes represent a concrete validation errors. Its method getMessage() returns some localized user-friendly message with a short explanation for a concrete validation error. E.g. the message could be set as an inline validation error with TextInputLayout.setError().

**Displaying invalidation reasons**

The validatable input fields do update their validation status on every input event, however the owner of the input fields must explicitly visualize the status. This can be done with the help of the OnErrorListener.

textField.setOnErrorListener { errorMessage: CharSequence? ->

// *Update the parent text input layout with the error message (null when valid)*

textInputLayout.*error* = errorMessage

}

**List of default validators**

For each input field class there is a validator class that should be assigned to the corresponding input field after layout inflation:

|  |  |
| --- | --- |
| **Input field** | **Default validator** |
| CardNumberEditText | DefaultCardNumberValidator |
| CardExpiryEditText | ExpiryDateValidator |
| CardSecurityCodeEditText | CvvValidator |
| CardHolderEditText | CardHolderValidator |
| CountryAutoCompleteTextView | DefaultCountryValidator |
| StreetEditText | FieldMaxLengthValidator |
| CityEditText |
| PostCodeEditText |

### Themes

The SDK UI can be easily customized by extending a Material theme and simply providing the theme ID to the SDK.   
  
There are 2 sets of themes that can be extended:

1. Geidea-branded themes – these themes contain Geidea-specific branding colors, rounded shapes and filled text input fields:

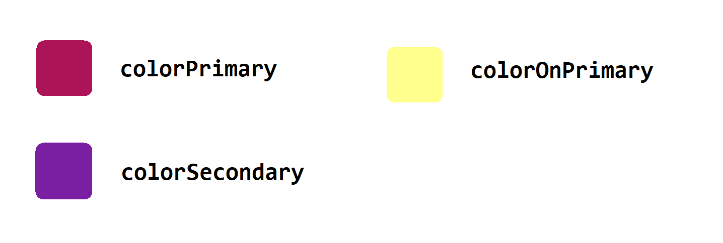
* **Gd.Theme.DayNight.NoActionBar** (**default**)
* **Gd.Theme.Material3.DayNight.NoActionBar (beta)**

1. Base (white-label) themes – these themes are as close as possible to the corresponding standard Material themes:

* **Gd.Base.Theme.NoActionBar**
* **Gd.Base.Theme.Light.NoActionBar**
* **Gd.Base.Theme.DayNight.NoActionBar**
* **Gd.Base.Theme.Material3.DayNight.NoActionBar (beta)**
* **Gd.Base.Theme.Material3.Light.NoActionBar (beta)**
* **Gd.Base.Theme.Material3.DynamicColors.DayNight (beta)**

Customizing the theme palette and placing a custom logo on the header:

<style name**="MyTheme"** parent**="Gd.Theme.DayNight.NoActionBar"**>  
 <item name**="colorPrimary"**>@color/pink</item>  
 <item name**="colorOnPrimary"**>@color/yellow</item>  
 <item name**="colorSecondary"**>@color/purple</item>  
  
 <item name**="gd\_merchantLogoImageViewStyle"**>@style/MyLogoImageViewStyle</item>  
</style>  
  
<style name**="MyLogoImageViewStyle"** parent**="@style/Gd.Base.Widget.ImageView.Logo.Merchant"**>  
 <item name**="android:layout\_width"**>wrap\_content</item>  
 <item name**="android:layout\_height"**>72dp</item>  
 <item name**="android:src"**>@drawable/ic\_my\_merchant\_logo</item>  
</style>

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

### Styles

Geidea Material components are normal Material components that can be styled with the standard approaches:

* through styles defined in an app-level or activity-level theme;
* through directly applying a style to the view definition in your XML;
* by supplying attributes programmatically through the constructor.  
    
  Some views are styled via custom style attributes defined in the theme. Be sure to use **Gd.Theme.\*** or **Gd.Base.Theme.\*** as these themes defines values of these style attributes pointing to default definitions. Such style attributes are:
* **gd\_merchantLogoImageViewStyle** – style of the ImageView centered inside the header displayed in most screens;
* **gd\_primaryButtonStyle** – style of a main Button which is displayed in most screens (Pay, Next, Proceed, Finalize, etc.)
* **gd\_secondaryButtonStyle** – style of a Button of non-major importance which is displayed in some screens (Cancel).
* **gd\_textInputExposedDropdownMenuStyle** – style of an exposed dropdown menu (e.g. Address country dropdown)

**Еxamples**  
See sdk-styles.xml in the Sample app.

### Simple payment form with PaymentFormView

PaymentFormView is a complete payment form component or more concretely - a vertical layout which has card and customer data input fields embedded inside and managed by itself.

**Layout hierarchy of PaymentFormView**

The following tree structure represents the public components in the form. Note: only the Geidea components are shown. The form contains more child views like card brand logos and “Pay” button.

* PaymentFormView
  + CardInputView
    - CardNumberEditText
    - CardExpiryDateEditText
    - CardSecurityCodeEditText
    - CardHolderEditText
  + EmailEditText – customer e-mail
  + AddressInputView – billing address
    - CountryAutoCompleteTextView – country drop-down selector
    - StreetEditText
    - CityEditText
    - PostEditText
  + AddressInputView – shipping address with the same child fields as billing address.

**Embedding in your XML layout**

Always wrap the form view in a scrollable container. You may need also a “Pay” or other UI which the customer will use to start the payment flow once the form is populated with valid data.

<ScrollView  
 android:id**="@+id/paymentFormScrollView"** android:layout\_width**="match\_parent"** android:layout\_height**="match\_parent"**>

<LinearLayout

android:layout\_width**="match\_parent"** android:layout\_height**="wrap\_content"**

android:orientation**="vertical"**>

<net.geidea.paymentsdk.ui.widget.PaymentFormView  
 android:id**="@+id/paymentFormView"** android:layout\_width**="match\_parent"** android:layout\_height**="wrap\_content"**/>

<Button

android:id**="@+id/payButton"**

android:layout\_width**="match\_parent"** android:layout\_height**="wrap\_content"**

android:text**="Pay"**/>

<LinearLayout/>

</ScrollView>

**Initializing**

Setup the payment form with your accepted card brands, supported countries, etc.

*paymentFormView.configure(GatewayApi.getMerchantConfiguration())*

**Populating**

First, let’s assume we have a val initial: PaymentIntent instance that holds the parameters we will prepopulate the payment form with. Prepopulating the form with it could look like this:

paymentFormView.showCustomerEmail = initial.showCustomerEmail  
if (initial.showCustomerEmail) {  
 paymentFormView.customerEmail = initial.customerEmail  
}  
  
paymentFormView.showAddress = initial.showAddress  
  
initial.billingAddress?.*let* **{** paymentFormView.billingAddress = **it }**initial.shippingAddress?.*let* **{** paymentFormView.shippingAddress = **it }**if (initial.billingAddress?.countryCode.*isNullOrEmpty*()) {  
 paymentFormView.billingAddressCountryCode = **"SAU"**}  
  
if (initial.shippingAddress?.countryCode.*isNullOrEmpty*()) {  
 paymentFormView.shippingAddressCountryCode = **"SAU"**}  
  
val same = paymentFormView.billingAddress.*equalsIgnoreCase*(paymentFormView.shippingAddress)  
paymentFormView.isSameAddressChecked = same || initial.shippingAddress.*isNullOrEmpty*()

**Reading form data and starting the payment flow**

*payButton.setOnClickListener { payButton ->  
 // Create a new “final” payment intent that contains the initial data combined with any data that the customer decided to override.  
 val finalPaymentIntent = PaymentIntent* **{** // Populate the merchant-related properties...  
 paymentOperation = initial.paymentOperation  
 amount = initial.amount  
 currency = initial.currency  
 merchantReferenceId = initial.merchantReferenceId  
 callbackUrl = initial.callbackUrl  
 showCustomerEmail = initial.showCustomerEmail  
 showAddress = initial.showAddress  
 cardOnFile = initial.cardOnFile  
 initiatedBy = initial.initiatedBy  
 agreementId = initial.agreementId  
 agreementType = initial.agreementType  
 bundle = initial.bundle  
  
 // ...then read and populate customer data from the form  
  
 val card: Card? = paymentFormView.card  
 paymentMethod = *PaymentMethod* **{** cardHolderName = card?.cardHolderName  
 cardNumber = card?.cardNumber  
  expiryDate = card?.expiryDate  
 cvv = card?.cvv  
 **}** customerEmail = if (paymentFormView.showCustomerEmail) {  
 paymentFormView.customerEmail  
 } else {  
 initial.customerEmail  
 }  
  
 billingAddress = if (paymentFormView.showAddress) {  
 paymentFormView.billingAddress  
 } else {  
  initial.billingAddress  
 }  
  
 shippingAddress = if (paymentFormView.showAddress) {  
  if (paymentFormView.isSameAddressChecked) {  
  paymentFormView.billingAddress  
 } else {  
  paymentFormView.shippingAddress  
 }  
 } else {  
 initial.shippingAddress  
 }  
 **}***)*

*// Start the Payment flow with the “final” payment intent*

paymentLauncher.launch(finalPaymentIntent) *}*

**Reading the entire content of the form**

The entire content can be read with val data: PaymentFormData.

**Getting / setting individual field values**

It is possible to modify and obtain values for specific fields only. Any changes will affect the respective input field value even if the field is not currently visible.

var card: Card?

var cardNumber: String?

var cardExpiryDate: String?

var cardSecurityCode: String?

var cardHolder: String?

var customerEmail: String?

var showCustomerEmail: Boolean

var showAddress: Boolean

var billingAddress: Address

var billingAddressCountryCode: String?

var billingAddressStreet: String?

var billingAddressCity: String?

var billingAddressPostCode: String?

var shippingAddress: Address

var shippingAddressCountryCode: String?

var shippingAddressStreet: String?

var shippingAddressCity: String?

var shippingAddressPostCode: String?

var isSameAddressChecked: Boolean

var acceptedCardBrands: Set<CardBrand>

var isValid: Boolean

**Enabling the “Pay” button once the form is validated**

paymentFormView.setOnValidationChangedListener { data: PaymentFormData?, valid: Boolean ->

payButton.*isEnabled* = valid

}

**View state saving / restoration**

PaymentFormView and its child fields preserve their entire state. For code example you may look at the SamplePaymentFormActivity in the Demo app.

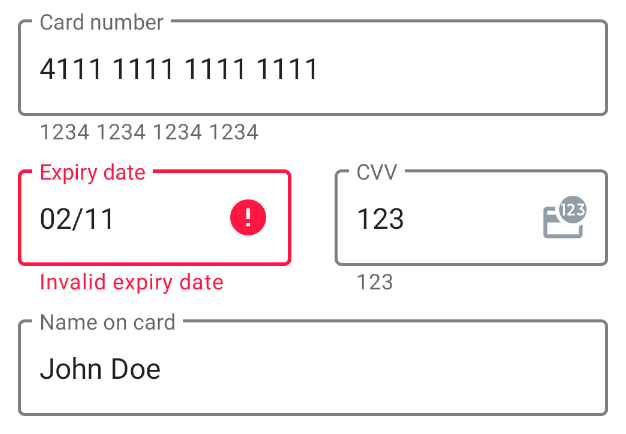
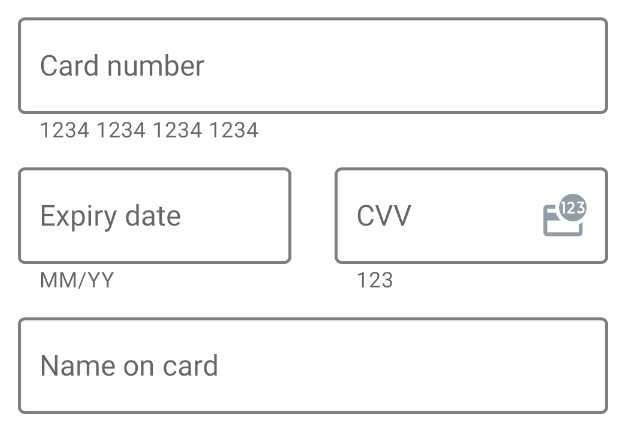
**Advanced use-cases**

If you need more flexibility and customizability than what PaymentFormView offers, then you could use Geidea’s Material components as building blocks. In this way you would have more control over the UI behavior and appearance. For more info see [Custom payment form by composing Geidea components](#_Custom_payment_form).

### Custom payment form by composing Geidea components

#### CardInputView

CardInputView is a layout composed of Material text input fields for the mandatory card data – card holder, number, expiry date and CVV. It’s a self-contained component that does all necessary validation, card brand detection, digits grouping, filtering of invalid characters and restricting to maximal allowed length.



**Embedding in your XML layout**

<net.geidea.paymentsdk.ui.widget.card.CardInputView  
 android:id**="@+id/cardInputView"** android:layout\_width**="match\_parent"** android:layout\_height**="wrap\_content"** />

**Getting the input data**

When the customer enters all required fields with valid input the card property becomes non-null.

val card: Card? = cardInputView.card

**Setting card number**

Partial, separated and non-separated numbers can be set.

cardInputView.setCardNumber(“4111 1111 1111 1111”) // or…

cardInputView.setCardNumber(“4111111111111111”)

The input field will automatically attempt to group the digits according to the grouping convention of the recognized card brand.

**Setting an expiry date**

The expiry date should be in 2-digit MONTH/YEAR format.

cardInputView.setCardExpiryDate(**"12/34"**)

**Setting a security code (CVV)**

The valid security code is 3- or 4-digit code depending on the card brand. It is 4 digits for *American Express* and 3 for all others**.**

cardInputView.setSecurityCode(**"123"**) // or…  
cardInputView.setSecurityCode(**"1234"**)

**Setting a card holder**

The card holder name must have a maximal length of 255 characters.

cardInputView.setCardHolder(**"John Doe"**)

**Checking for validity**

When Customer enters all required fields with valid input the value of isValid property becomes true.

val valid: Boolean = cardInputView.isValid

**Getting the currently recognized card brand**

val cardBrand: CardBrand = cardInputView.cardBrand  
when (cardBrand) {  
 CardBrand.Visa,  
 CardBrand.Mastercard,  
 CardBrand.AmericanExpress,  
 CardBrand.Mada -> {} // Recognized brand  
 CardBrand.Unknown -> {} // Unrecognized or invalid brand  
}

**Filtering only card brands that you accept**

By default, CardInputView accepts all card brands supported by Geidea. It must be configured in order to filter the non-accepted.

val merchantConfig = GatewayApi.getMerchantConfiguration()

val acceptedBrands= (merchantConfig.paymentMethods ?: *emptyList*())  
 .*map*(CardBrand::fromBrandName)  
 .*toSet*()  
 .*minus*(CardBrand.Unknown)

cardInputView.setCardBrandFilter(acceptedBrands::contains)

**Setting input listener**

cardInputView.setCardInputListener(object : CardInputView.CardInputAdapter() {  
 override fun onFocusChange(focusField: CardFieldType) {  
 // Implementation is left for sub-classes  
 }  
  
 override fun onFieldValidStatus(field: CardFieldType) {  
 // Implementation is left for sub-classes  
 }  
  
 override fun onFieldInvalidStatus(field: CardFieldType, status: ValidationStatus.Invalid) {  
 // Implementation is left for sub-classes  
 }

override fun onCardValidationChanged(cardValid: Boolean) {  
 payButton.*isEnabled* = cardValid}  
  
 override fun onCardInputComplete() {clearFocusAndDismissKeyboard()}  
})

**Using the card data to pay with**

Here is a minimalistic example of a function that initiates a payment with the collected card data

private fun payWith(card: Card) {

val paymentIntent = *PaymentIntent* **{** amount = **"**123.45**"**.toBigDecimal()  
 currency = **"**SAU**"**

**// Supply the card data to SDK to perform the required authentication / transactions**

paymentMethod = *PaymentMethod* **{** cardNumber = card.cardNumber  
 cardHolderName = card.cardHolderName  
 expiryDate = card.expiryDate  
 cvv = card.cvv  
  **}**

// More *PaymentIntent* parameters could be set here

}  
 paymentLauncher.launch(paymentIntent)

clearFocusAndDismissKeyboard()

}

**Setting card brand listener**

cardInputView.setOnCardBrandChangedListener **{** brand **->**

**// Highlight the icon of the card brand**

**}**

**Setting text watchers**

fun setCardHolderTextWatcher(cardHolderTextWatcher: TextWatcher)  
fun setCardNumberTextWatcher(cardNumberTextWatcher: TextWatcher)  
fun setExpiryDateTextWatcher(expiryDateTextWatcher: TextWatcher)  
fun setSecurityCodeTextWatcher(securityCodeTextWatcher: TextWatcher)

For ease-of-use the abstract TextWatcherAdapter could be extended which has empty implementations of the abstract methods.

**Setting security code (CVV) icon click listener**

fun setSecurityCodeEndIconClickListener(listener: OnClickListener?)

#### AddressInputView

AddressInputView is a layout composed of text fields for entering Address data. It can be used for collecting billing and shipping addresses.

**Embedding in your XML layout**

<net.geidea.paymentsdk.ui.widget.address.AddressInputView  
 android:id**="@+id/addressInputView"** android:layout\_width**="match\_parent"** android:layout\_height**="wrap\_content"** />

**Getting / setting input data**

**The whole address**

var address: Address = addressInputView.address

addressInputView.address = Address { /\*…\*/ }

**Country code**

addressInputView.countryCode = “SAU”

val countryCode: String? = addressInputView.countryCode

**Street**

addressInputView.street = “Wall st. 11”

val street: String? = addressInputView.street

**City**

addressInputView.city = “New York”

val city: String? = addressInputView.city

**Postcode**

addressInputView.postCode = “10005”

val postCode: String? = addressInputView.postCode

**Checking for validity**

When all populated address fields are valid then isValid property becomes true.

val valid: Boolean = addresInputView.isValid

**Setting input events listener**

addresInputView.setAddressInputListener(object : AddressInputAdapter() {

fun onFocusChange(focusField: AddressFieldType) {}  
 fun onFieldValidStatus(field: AddressFieldType) {}  
 fun onFieldInvalidStatus(field: AddressFieldType, status: ValidationStatus.Invalid) {}  
 fun onAddressValidationChanged(address: Address, valid: Boolean) {}

)

### Custom payment form with non-Geidea components

It is possible to use any kind of UI components to collect the payment data. The requirement is to populate a PaymentIntent instance and its paymentMethod property then [launch the Payment flow](#_Payment_flow) with that instance. See SamplePaymentFormActivity in the Demo app.

### Custom integration

Geidea Android SDK provides the possibility for totally seamless integration. This way is intended for complex use-cases and requires significantly more effort that the other integration types (**Simple** and **Mixed)**. Typically, you will need to set-up the UI - a payment form and WebView for authentication, then implement correct transaction flows with the help of the [API](#_API) methods.

## Code samples

Code samples can be found in the Demo app. They demonstrate different types of integration of the SDK:

**SampleSimplestPaymentActivity**

Sample with the minimally required code to perform a payment.

**~~SampleThemeActivity~~**

~~How to customize the Payment flow via customized theme.~~ Since version 3.0 theme switching is moved to MainActivity. See the “Theme” drop-down.

**SamplePaymentFormActivity**

How to embed a payment form in your XML layout, populate and read data, listen for events and start payment flow. Uses customized theme.

**SampleAddressActivity**

How to embed an Address input view in your XML, populate and read data, listen for events and manipulate the view.

**SampleAddressFieldsActivity**

How to embed individual Address input fields in your XML, populate and read data, listen for validation events.

**SampleCardActivity**

How to embed an Address input view in your XML, populate and read data, listen for events and manipulate the view.

**SampleCardFieldsActivity**

How to embed individual Card input fields in your XML, populate and read data, listen for validation events.

**SampleEInvoiceActivity**

How to perform CRUD operations with e-Invoices.

**SampleOrdersActivity**

How to get and search by status and date orders. How to get specific order by id.

Appendix A. Glossary

|  |  |
| --- | --- |
| Glossary Item | Description |
| SDK | Software Development Kit |
| 3DS | 3D Secure - secure protocol designed to ensure enhanced security and strong authentication |
| MPGS | Mastercard Payment Gateway Services |
| BIN | Bank Identification Number - the initial four to six numbers that appear on a credit/debit card |
| HPP | Geidea Hosted Payment Page |
| BNPL | Buy Now Pay Later (Installments) |