Android Card Management SDK - Network International

Integration Guide Document – Version 2.4

**Version History**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Version | Date | Author | Description of changes | | 0.1 | 22.11.2022 | Cosmin Telescu | Initial Draft | | 0.2 | 22.11.2022 | Ciprian Salomir | Second draft | | 0.3 | 30.11.2022 | Cosmin Telescu | Updated subsection 4.1🡪 added note | | 1.0 | 7.12.2022 | Ciprian Salomir | Sections 4.3 and 4.4.2 were added | | 1.1 | 23.12.2022 | Cosmin Telescu | Replaced section 4 with new section | | 2.0 | 15.05.2023 | Ciprian Salomir | All changes updated | | 2.1 | 25.05.2023 | Ciprian Salomir | Client sugestions update | | 2.2 | 14.06.2023 | Tudor Stoicovici | Small content arrangements and formatting | | 2.3 | 29.09.2023 | Daniel Sebastian Lazea | Updated subsections 3.2.5.2 and 3.5.4 | | 2.4 | 06.12.2023 | Daniel Sebastian Lazea | Added section 6 | |  |  |  |  | |

Contents

[1 Introduction 3](#_Toc137726830)

[1.1 Document Purpose 3](#_Toc137726831)

[1.2 SDK Overview 3](#_Toc137726832)

[1.3 Quick Start 3](#_Toc137726833)

[2 Android SDK Integration 4](#_Toc137726834)

[2.1 SDK Requirements 4](#_Toc137726835)

[2.2 Public Maven Repository 4](#_Toc137726836)

[3 Using the SDK 6](#_Toc137726837)

[3.1 How to Use 6](#_Toc137726838)

[3.2 Expected Input Parameters – NIInput Object 6](#_Toc137726839)

[3.2.1 Bank code 6](#_Toc137726840)

[3.2.2 Card Identifier Id 6](#_Toc137726841)

[3.2.3 Card Identifier Type 6](#_Toc137726842)

[3.2.4 Connection Properties 7](#_Toc137726843)

[3.2.5 Display attributes 7](#_Toc137726844)

[3.3 Form Factory Interface 11](#_Toc137726845)

[3.4 Programatic Interface 12](#_Toc137726846)

[3.5 How to use the SDK Components 13](#_Toc137726847)

[3.5.1 Displaying the Card Details Form 13](#_Toc137726848)

[3.5.2 Displaying Card Details Using the Card Fragment component from an Activity 15](#_Toc137726849)

[3.5.3 Displaying Card Details Using the Card Fragment component from another Fragment 17](#_Toc137726850)

[3.5.4 Displaying Set/Verify/Change PIN component from an Activity 20](#_Toc137726851)

[3.5.5 Displaying Set/Verify/Change PIN component from another Fragment 24](#_Toc137726852)

[4 The SDK User Interface 27](#_Toc137726853)

[4.1 Display Card Details UI 27](#_Toc137726854)

[4.2 Set PIN Component UI 33](#_Toc137726855)

[4.3 Verify PIN Component UI 34](#_Toc137726856)

[4.4 Change PIN Component UI 35](#_Toc137726857)

[4.5 Localization of the SDK (Current Language Support) 36](#_Toc137726858)

[4.6 Light / Dark mode UI 38](#_Toc137726859)

[5 Sample App 39](#_Toc137726860)

6 SDK Errors ………………………………………………………………………………………………40

**6.1 Invalid PIN Error ……………………………………………………………………………………40**

**6.2 Specific Errors ………………………………………………………………………………….…..40**

# Introduction

## Document Purpose

The purpose of the Android SDK Integration guide is to serve as a technical documentation for customers that want to integrate with Network International APIs into their mobile applications.

## SDK Overview

The Network International Android Card Management SDK allows you to quickly display card details, and integrate functionalities like set pin, verify pin, and change pin, into your Android app.

Network International APIs offer the following operations:

* Display the card details in a fragment component, activity, which contains Clear Card Number (PAN), Expiry Date and Clear CVV2
* Display a PIN-pad to capture and confirm/set the PIN of the card
* Display a PIN-pad to capture the old PIN and new PIN and confirm/change the PIN of the card
* Display a PIN-pad to capture and verify the PIN of the card
* Retrieve card details programmatically
* Set PIN programmatically
* Change PIN programmatically
* Verify PIN programmatically

## Quick Start

The links below will quickly guide the developer through the most common way of integrating the SDK. For any other option of integration, please check the other sub-chapters as well.

* **Public Maven Repository**
* [**Expected Input Parameters – NIInput Object**](#_Expected_Input_Parameters)
* [**Displaying Card Details Using the Card Fragment component from another Fragment**](#_Displaying_Card_Details)
* [**Displaying Set/Verify/Change PIN component from another Fragment**](#_Displaying_Set/Verify/Change_PIN)

# Android SDK Integration

The SDK integration can be done in a user friendly manner using Gradle to import the library from the public repository.

## SDK Requirements

* Minimum Android Version: 5.0 (API version 21)
* Maximum Target SDK Android API version: 31
* Recommended IDE version: Android Studio Dolphin | 2021.3.1 Patch 1 or newer
* Recommended Kotlin version: 1.7.20 or higher

## Public Maven Repository

The SDK library is published on the Jitpack repository (jitpack.io) and can be found at the following URL :

<https://jitpack.io/#network-international/card-management-sdk-android/>.

From the user perspective there are two steps involved in the process of importing the library into the Android project:

1. Add the Jitpack repository to your repositories list in your settings.gradle file:

A screenshot of a computer program

Description automatically generated with medium confidence

Please add the following line of code:

maven **{** url 'https://jitpack.io' **}**

1. Update your app module build.gradle file to include the library as a dependency:

A screenshot of a computer program

Description automatically generated with medium confidence

Please add the following lines of code:

// NI SDK from Jitpack repository  
implementation 'com.github.network-international:card-management-sdk-android:1.0.34'

Please note the version number used above is only for sample. Please consult our github and jitpack to know our latest version.

Hit the “Sync Now” button on the top right of the screen and wait for the import to be completed. Once the sync is done, the SDK is added to your project.

# Using the SDK

## How to Use

The SDK offers two interfaces, Form Factory and Programatic Interface, which declare the public API. Each interface has a corresponding class which offer the implementation of the interface.

In addition to this the SDK offers Card Details and Verify/Set/Change PIN components which can be accessed directly from your mobile app code.

## Expected Input Parameters – NIInput Object

To use our SDK functionality, the following input is expected:

* Bank Code
* Card Identifier Id
* Card Identifier Type
* Connection Properties
* Display Attributes

data class NIInput(  
 val bankCode: String,  
 val cardIdentifierId: String,  
 val cardIdentifierType: String,  
 val connectionProperties: NIConnectionProperties,  
 val displayAttributes: NIDisplayAttributes? = null  
) : Serializable

### Bank code

* Represents the unique identifier of your financial institution which is initially provided by Network International during onboarding project. This parameter will be passed in all API integration.
* Required parameter

### Card Identifier Id

* Represents the ID of the card
* Required parameter

### Card Identifier Type

* Represents the type of identifier passed.
* Required parameter

Types supported:

* **EXID** (External ID)
* **CONTACT\_NUMBER** (Clear PAN)

### Connection Properties

* This set of parameters is used to regulate the connectivity between the SDK and your proxy middleware as per the architecture diagram provided by Network International.
* Required parameter
* Connection Properties:
  + rootUrl: Base endpoint of your middleware upon which our URL route will be concatenated. The URL can be specified with or without ending slash character i.e. : <https://apitest.network.ae/> or <https://apitest.network.ae>
  + token: Authentication token for your own middleware. This token will be passed in every API call in the header (Authorization) to ensure you perform validation of the identity of the caller.

data class NIConnectionProperties(  
 val rootUrl: String,  
 val token: String  
) : Serializable

### Display attributes

* This set of parameters allows for UI customization
* Display attributes parameter is optional. You can set one or more attributes, or even none of them.

Display attributes offered are:

* Fonts (fonts)
* Card Attributes (background image, hide/show property, text positioning) (cardAttributes)
* setPinMessageAttributes, verifyPinMessageAttributes, changePinMessageAttributes are attributes to control the UI layout for Set, Verify or Change PIN component, respectively. If set these parameters enable the displaying of a custom success/error layout as described by the PinMessageAttributes class parameters.
* theme enables the option to force light or dark mode if required
* language enables the option to force English or Arabic language (for now) if required

data class NIDisplayAttributes(  
 // this parameter is optional  
 // if set, these fonts will be used in the UI forms; if not set will use default fonts  
 val fonts: List<NIFontLabelPair>? = null,  
  
 // this parameter is optional  
 // if set, the card details will take into account the attributes passed into this variable  
 // if not set, will take the default values  
 val cardAttributes: NICardAttributes? = null,  
  
 // the next three parameter are optional  
 // if set an custom layout will be displayed on completion, and the component will navigate  
 // back on "doneButton" specified as an @IdRes val buttonResId: Int  
 val setPinMessageAttributes: PinMessageAttributes? = null,  
 val verifyPinMessageAttributes: PinMessageAttributes? = null,  
 val changePinMessageAttributes: PinMessageAttributes? = null,  
  
 // This parameter is optional.  
 // If not set the SDK will follow your parent app day/night mode based on OS settings or as requested by your app.  
 // The recommended way for using this parameter is to leave it unset, unless you have some special requirements.  
 // If a value is set, the SDK will emulate (force) day/night mode, regardless of the system settings.  
 val theme: NITheme? = null,

// This parameter is optional.  
 // If not set, the SDK will automatically choose English as default language.  
 // If a value is set, the SDK will force English/Arabic language.

val language: NILanguage? = null  
): Serializable

enum class NITheme: Serializable {  
 *LIGHT*, *DARK*}

#### Theme

SDK is offering support for Day/Night mode which is managed automatically by the operating system OS or requested internally by your app.

The theme parameter is optional.

If not set the SDK will follow your parent app day/night mode based on OS settings or as requested by your app.

The recommended way is to leave this parameter unset, unless you have some special requirements.

If a value is set, the SDK will emulate (force) day/night mode, regardless of the system settings.

#### Language

The Language parameter is optional and the options are either English, or Arabic languages.

If not set, the SDK will follow English by default,

If a value is set, the SDL will emulate (force) the chosen language, regardless of the system settings. The emulated language will only be applied on the SDK Views, while the rest of the app will remain unchanged.

Language is a list of NILanguage objects, consisting of the following:

enum class NILanguage : Serializable {  
 *ENGLISH*, *ARABIC*}

#### Fonts

- fonts parameter is a list of NIFontLabelPair objects providing the option to have custom text size and fonts to the following UI resources:

- Card Details component:

*CARD\_NUMBER\_LABEL*,  
 *CARD\_NUMBER\_VALUE\_LABEL*,  
 *EXPIRY\_DATE\_LABEL*,  
 *EXPIRY\_DATE\_VALUE\_LABEL*,  
 *CVV\_LABEL*,  
 *CVV\_VALUE\_LABEL*,  
 *CARD\_HOLDER\_NAME\_LABEL*

data class NIFontLabelPair(  
 val uiFont: UIFont,  
 var label: NILabels  
): Serializable  
  
data class UIFont(  
 @FontRes  
 val fontRes: Int? = null,  
 // interpreted as "scaled pixel" units Sp  
 val textSize: Int  
): Serializable  
  
enum class NILabels: Serializable {  
  
 // Card Details  
 *CARD\_NUMBER\_LABEL*,  
 *CARD\_NUMBER\_VALUE\_LABEL*,  
 *EXPIRY\_DATE\_LABEL*,  
 *EXPIRY\_DATE\_VALUE\_LABEL*,  
 *CVV\_LABEL*,  
 *CVV\_VALUE\_LABEL*,  
 *CARD\_HOLDER\_NAME\_LABEL*}

The NIFontLabelPair class describe a pair of NILabels and UIFont objects in order to customize certain text elements of the UI compoenents.

The UIFont class specify the font resource fontRes and textSize interpreted as "scaled pixel" units Sp.

#### Card Attribute

data class NICardAttributes(  
 // if true, the card details will be hidden/masked by default; if false, the card details will be visible by default  
 val shouldHide: Boolean = true,  
 // if set, this image will be used as background for the card details view; if not set, it will use default image from sdk  
 @DrawableRes  
 val backgroundImage: Int? = null,  
 // if set will apply new text positioning values for Card Details components: cardNumberLine, dateCvvLine and cardHolderNameLine  
 // if not set default positioning will be used  
 val textPositioning: TextPositioning? = null  
): Serializable  
  
// the ratio of the parent container width/height in the range 0..1 (meaning 0 to 100% percent)  
// relative to the left|top corner (which is the axis origin point)  
// leftAlignment: all three groups of views (lines) have the same left alignment  
// cardNumberGroupTopAlignment: card number line top alignment  
// dateCvvGroupTopAlignment: date cvv line top alignment  
// cardHolderNameGroupTopAlignment: card holder name line top alignment  
data class TextPositioning(  
 val leftAlignment: Float? = null,  
 val cardNumberGroupTopAlignment: Float? = null,  
 val dateCvvGroupTopAlignment: Float? = null,  
 val cardHolderNameGroupTopAlignment: Float? = null  
): Serializable

The NICardAttributes class is described by the following parameters:

- shouldHide if true, the card details will be hidden/masked by default; if false, the card details will be visible by default

- backgroundImage if set, this image will be used as background for the card details view; if not set, it will use default image from sdk

- textPositioning if set will apply new text positioning values for Card Details components: cardNumberLine, dateCvvLine and cardHolderNameLine, if not set default positioning will be used

The TextPositioning class is used to set new values for the left and top position of the card number line, date cvv line and card holder line (group of views).

It is described by the following parameters:

- leftAlignment all three groups of views (lines) have the same left alignment.

- cardNumberGroupTopAlignment card number line top alignment

- dateCvvGroupTopAlignment date cvv line top alignment

- cardHolderNameGroupTopAlignment card holder name line top alignment

All values represent the ratio of the parent container width/height in the range 0..1 (meaning 0 to 100% percent).

The left and top alignment are interpreted as a percent so the groups position will scale along with the view.

These parameters are optional and if not set default positioning will be used, otherwise the new values will override the existing default positions.

#### PIN Message Attributes

data class PinMessageAttributes(  
 val successAttributes: SuccessErrorScreenAttributes,  
 val errorAttributes: SuccessErrorScreenAttributes  
): Serializable  
  
data class SuccessErrorScreenAttributes(  
 @LayoutRes val layoutId: Int,  
 @IdRes val buttonResId: Int  
): Serializable

The PinMessageAttributes class have two properties of type SuccessErrorScreenAttributes one intended for the happy flow and the other for the error scenario.

It give us the opportunity to set a custom success/error screen:

- successAttributes : success custom layout message

- errorAttributes : error custom layout message

All parameters are required.

The SuccessErrorScreenAttributes data class is described by the following parameters:

- layoutId : represent the layout resource id of the custom layout intended to be used on the success/error screen

- buttonResId : the “Done” button resource id (should be part of the specfied layout)

All parameters are required.

## Form Factory Interface

The public interface Form Factory is the component which is asociated with a user interface UI :

interface NICardManagementFormsAPI {  
  
 fun displayCardDetailsForm(  
 input: NIInput  
 )

}

The implementation of the interface is NICardManagementForms class:

typealias OnSuccessErrorCancelCompletion = (isSuccess: NISuccessResponse?, isError: NIErrorResponse?, isUserCanceled: Boolean) -> Unit

class NICardManagementForms(  
 private val activity: ComponentActivity,  
 displayCardDetailsOnCompletion: OnSuccessErrorCancelCompletion = **{** \_, \_, \_ **-> }**,  
 setPinOnCompletion: OnSuccessErrorCancelCompletion = **{** \_, \_, \_ **-> }**,  
 changePinOnCompletion: OnSuccessErrorCancelCompletion = **{** \_, \_, \_ **-> }**,  
) : NICardManagementFormsAPI {

override fun displayCardDetailsForm(  
 input: NIInput  
 ) {

// implementation  
 }

}

The form factory part of the SDK API is intended to be called from an Activity in the client app and will result is displaying a new activity with an Card Details component inside along with the back navigation buton. NICardManagementFormsAPI is made as a distinct part from the programatic component NICardManagementAPI which is intended to be used from the ViewModel side of the client app (injected as an external dependency in the ViewModel constructor).

Using this Form Factory interface allows the consumer to not be in the PCI-DSS scope as all sensitive information are managed internally and not accessible from the rest of the application.

## Programatic Interface

The programmatic interface is intended to be used in the event of a custom UI requirements and offers all the functionality of the SDK, returning the data but without involving any UI.

**IMPORTANT:**

Please keep in mind that using the programmatic interface will pull you back into the PCI-DSS scope as you will handle sensitive information such as:

* Clear Card Number
* Clear CVV
* Clear PIN

We strongly suggest informing Network International if you are planning to use this integration method.

interface NICardManagementAPI {  
  
 suspend fun getCardDetails(  
 input: NIInput  
 ): DetailsErrorResponse  
  
 suspend fun setPin(  
 pin: String,  
 input: NIInput  
 ): SuccessErrorResponse  
  
 suspend fun verifyPin(  
 pin: String,  
 input: NIInput  
 ): SuccessErrorResponse  
  
 suspend fun changePin(  
 oldPin: String,  
 newPin: String,  
 input: NIInput  
 ): SuccessErrorResponse  
}  
  
data class SuccessErrorResponse(  
 val isSuccess: NISuccessResponse?,  
 val isError: NIErrorResponse?  
)  
  
data class DetailsErrorResponse(  
 val details: NICardDetailsResponse?,  
 val isError: NIErrorResponse?  
)

The implementation of the interface:

object NICardManagement : NICardManagementAPI {  
  
 override suspend fun getCardDetails(  
 input: NIInput  
 ): DetailsErrorResponse {  
 // implementation}

override suspend fun setPin(  
 pin: String,  
 input: NIInput  
 ): SuccessErrorResponse {  
 // implementation}  
  
 override suspend fun verifyPin (  
 pin: String,  
 input: NIInput  
 ): SuccessErrorResponse {  
 // implementation}  
  
 override suspend fun changePin(  
 oldPin: String,  
 newPin: String,  
 input: NIInput  
 ): SuccessErrorResponse {  
 // implementation}  
  
}

## How to use the SDK Components

### Displaying the Card Details Form

Card Details form is implemented as an Activity which is displayed on the screen when you call the SDK specific method displayCardDetailsForm(niInput):

class MainActivity : AppCompatActivity() {  
  
 private lateinit var binding: ActivityMainBinding  
 private lateinit var viewModel: MainViewModel  
 private val niInput: NIInput  
 get() = viewModel.makeInputObject()  
 private val pinLength: NIPinFormType  
 get() = viewModel.getPINLength()  
  
 private val niCardManagementForms = NICardManagementForms(  
 this,  
 displayCardDetailsOnCompletion = getCompletionHandler("displayCardDetailsForm")  
 )  
  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 setArchitectureComponents()  
 initializeUI()  
 setViewModelData()  
 }  
  
 private fun setArchitectureComponents() {  
 viewModel = ViewModelProvider(this)[MainViewModel::class.*java*]  
 binding = DataBindingUtil.setContentView(this, R.layout.*activity\_main*)  
 binding.*lifecycleOwner* = this  
 binding.*viewModel* = viewModel  
 }  
  
 private fun initializeUI() {  
 binding.*apply* **{** recyclerView.*apply* **{** *layoutManager* = LinearLayoutManager(*context*)  
 *adapter* = EntriesListAdapter()  
 setHasFixedSize(true)  
  
 this@MainActivity.viewModel.entriesItemsLiveData.observe(this@MainActivity) **{** itemModels **->** itemModels?.*let* **{** (*adapter* as EntriesListAdapter).setItems(**it**)  
 **}  
 }  
 }** cardDetailsButton.setOnClickListener **{** niCardManagementForms.displayCardDetailsForm(niInput)  
 **}  
 }** }  
  
 private fun setViewModelData() {  
 if (viewModel.entriesItemModels.isEmpty()) {  
 val entries = *listOf*(  
 EntriesItemModel(*BANK\_CODE*, getString(R.string.*bank\_code\_txt*), "EAND"),  
 EntriesItemModel(*CARD\_ID*, getString(R.string.*card\_identifier\_id\_txt*), "52913582150735206039"),  
 EntriesItemModel(*CARD\_TYPE*, getString(R.string.*card\_identifier\_type\_txt*), "EXID"),  
 EntriesItemModel(*ROOT\_URL*, getString(R.string.*root\_url\_txt*), "https://apitest.network.ae"),  
 EntriesItemModel(*TOKEN*, getString(R.string.*token\_txt*), "ybp4s3utm9grex68r765rh2s"),  
 EntriesItemModel(*PIN\_LENGTH*, getString(R.string.*pin\_length\_txt*), NIPinFormType.*FOUR\_DIGITS*.name, getString(R.string.*pin\_length\_placeholder*))  
 )  
 viewModel.setEntriesItems(entries)  
 }  
 }

private fun getCompletionHandler(formName: String): OnSuccessErrorCancelCompletion =  
 **{** success, error, canceled **->** if (canceled) {  
 Log.d(TAG, "$formName canceled by the user")  
 } else {  
 success?.*let* **{** Log.d(TAG, "$formName OK")  
 **}** error?.*let* **{** Log.d(TAG, "$formName execution has error")  
 **}** }  
 **}** companion object {  
 const val TAG = "MainActivity"  
 }

The instance of the class NICardManagementForms() should be created before the onCreate() method call. This **must** be called **unconditionally**, as part of initialization path, typically as a field initializer of an Activity. *This rule is enforced by the Android SDK API*.

private val niCardManagementForms = NICardManagementForms(  
 this,  
 displayCardDetailsOnCompletion = getCompletionHandler("displayCardDetailsForm")  
)

The first parameter of the constructor is mandatory and represent the current activity, and the second is optional and represent a completion handler callback implementation for the Card Details compoment.

Then in onCreate() you can call the SDK public method for displaying the Card Details component activity:

cardDetailsButton.setOnClickListener **{** niCardManagementForms.displayCardDetailsForm(niInput)  
**}**

### Displaying Card Details Using the Card Fragment component from an Activity

Let’s create a new activity for demo purpose called **CardUsageDemoActivity**. In order to use the custom component **CardDetailsFragment** our activity should represent an implementation to the **CardDetailsFragment.OnFragmentInteractionListener** interface**.** This step is mandatory in order to be able to receive the operation result from the fragment component. This is step is required and the SDK will throw an RuntimeException if not implemented.

class CardUsageDemoActivity : AppCompatActivity(), CardDetailsFragment.OnFragmentInteractionListener {  
  
 lateinit var niInput: NIInput  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_card\_usage\_demo*)  
 getDataFromBundle()  
 initializeUI()  
 }  
  
 private fun getDataFromBundle() {  
 *intent*.*getSerializableExtraCompat*(Extra.EXTRA\_NI\_INPUT, NIInput::class.*java*)?.*let* **{** niInput = **it  
 }** ?: throw RuntimeException("${this::class.*java*.*simpleName*} intent serializable ${Extra.EXTRA\_NI\_INPUT} is missing")  
 }  
  
 private fun initializeUI() {  
 val cardDetailsFragment = CardDetailsFragment.newInstance(niInput)  
 *supportFragmentManager*.beginTransaction().*apply* **{** add(R.id.*card\_container*, cardDetailsFragment, CardDetailsFragment.TAG)  
 commit()  
 **}** }  
  
 override fun onCardDetailsFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(TAG, **it**.message)  
 **}** response.isError?.*let* **{** Log.d(TAG, **it**.errorMessage)  
 **}** }  
  
 companion object {  
 const val TAG = "CardUsageDemoActivity"  
 }  
}

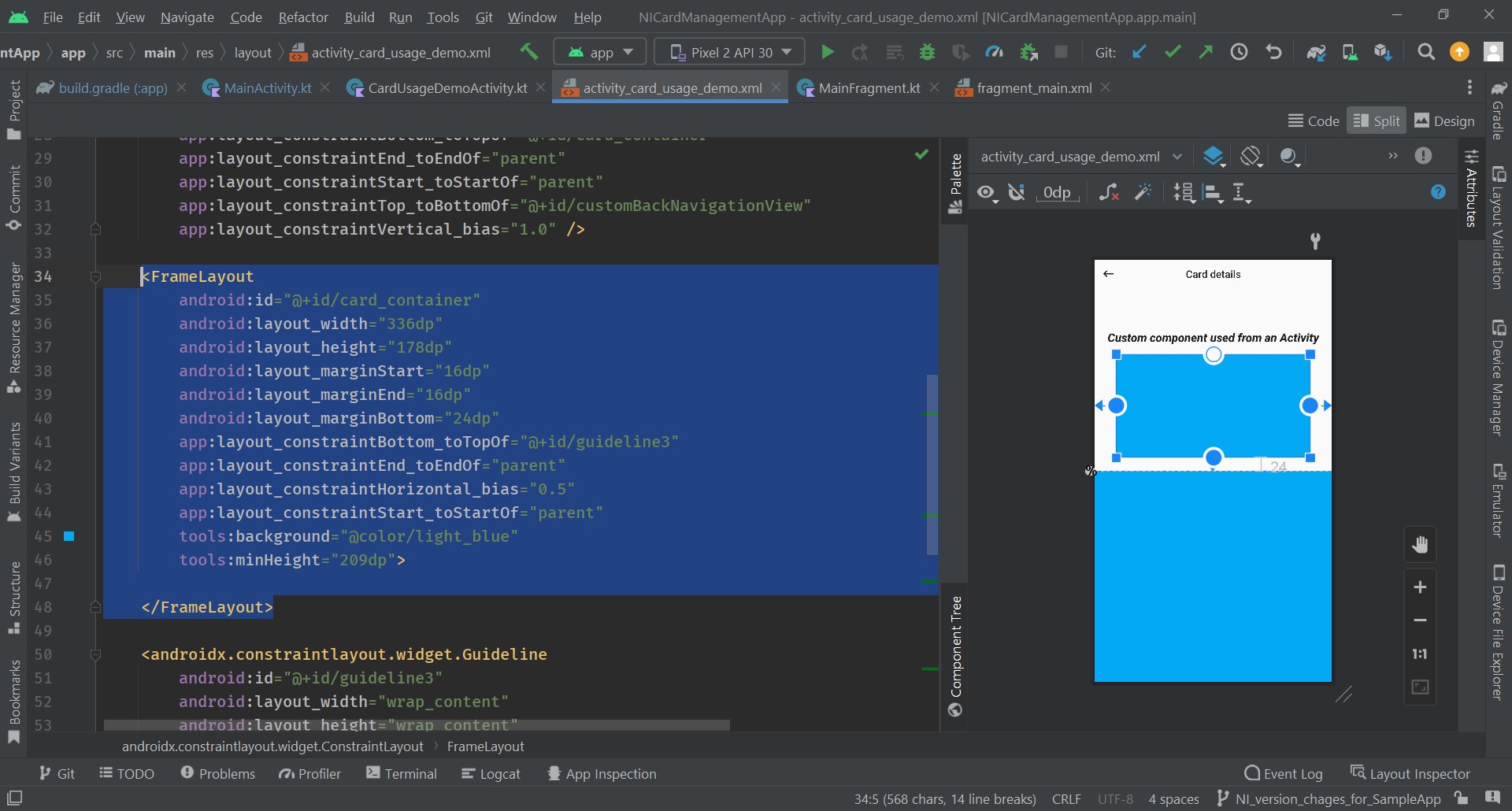
The UI of the activity should offer a container (can be a FrameLayout) described here by **R.id.card\_container,** which have the appropiate width and height dimension (i.e. width = 336dp and height=210dp). The fragment component will adjust itself to the size of the container. The width and height of the container is recommended to keep the aspect ratio of 16:10 (width:height), which guarantee a good user experience related to card UI.

Using a transaction on the supportFragmentManager, as described by the above highlighted code, we add the Card Details Fragment component to our CardUsageDemoActivity UI.

The layout of the demo activity:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".CardUsageDemoActivity">  
  
 …  
  
 <FrameLayout  
 android:id="@+id/card\_container"  
 android:layout\_width="336dp"  
 android:layout\_height="178dp"  
 android:layout\_marginStart="16dp"  
 android:layout\_marginEnd="16dp"  
 android:layout\_marginBottom="24dp"  
 app:layout\_constraintBottom\_toTopOf="@+id/guideline3"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 tools:background="@color/light\_blue"  
 tools:minHeight="209dp">  
  
 </FrameLayout>

…  
  
  
</androidx.constraintlayout.widget.ConstraintLayout>



The Activity will receive a callback on completion of the card component operation described by the SuccessErrorResponse object:

override fun onCardDetailsFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(TAG, **it**.message)  
 **}** response.isError?.*let* **{** Log.d(TAG, **it**.errorMessage)  
 **}**}

### Displaying Card Details Using the Card Fragment component from another Fragment

The first step is to implement the CardDetailsFragmentBase.OnFragmentInteractionListener interface, in order to be able to receive the operation result from the fragment component. This is step is required and the SDK will throw an RuntimeException if not implemented.

class MainFragment :  
 Fragment(),  
 CardDetailsFragmentBase.OnFragmentInteractionListener {  
  
 private var \_binding: FragmentMainBinding? = null  
 private val binding: FragmentMainBinding  
 get() = \_binding!!  
  
 private lateinit var niInput: NIInput  
 private lateinit var niPinFormType: NIPinFormType  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *arguments*?.*getSerializableCompat*<NIInput>(Extra.EXTRA\_NI\_INPUT)?.*let* **{** niInput = **it  
 }** ?: throw RuntimeException("${this::class.*java*.*simpleName*} arguments serializable ${Extra.EXTRA\_NI\_INPUT} is missing")  
  
 *arguments*?.*getSerializableCompat*<NIPinFormType>(Extra.EXTRA\_NI\_PIN\_FORM\_TYPE)?.*let* **{** niPinFormType = **it  
 }** ?: throw RuntimeException("${this::class.*java*.*simpleName*} intent serializable ${Extra.EXTRA\_NI\_PIN\_FORM\_TYPE} is missing")  
 }  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?): View? {  
 // Inflate the layout for this fragment  
 \_binding = DataBindingUtil.inflate(inflater, R.layout.*fragment\_main*, container, false)  
 return binding.*root* }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 val cardDetailsFragment = CardDetailsFragmentFromFragment.newInstance(niInput)  
 *childFragmentManager*.beginTransaction().*apply* **{** add(R.id.*fragment\_card\_container*, cardDetailsFragment, CardDetailsFragmentFromFragment.TAG)  
 commit()  
 **}** }  
  
 override fun onDestroyView() {  
 super.onDestroyView()  
 \_binding = null  
 }  
  
 override fun onCardDetailsFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(MainActivity.TAG, "MainFragment ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(MainActivity.TAG, "MainFragment ${**it**.error} ${**it**.errorMessage}")  
 **}** }  
  
 companion object {  
 @JvmStatic  
 fun newInstance(input: NIInput, type: NIPinFormType) = MainFragment().*apply* **{** *arguments* = Bundle().*apply* **{** putSerializable(Extra.EXTRA\_NI\_INPUT, input)  
 putSerializable(Extra.EXTRA\_NI\_PIN\_FORM\_TYPE, type)  
 **}  
 }** const val TAG = "MainFragment"  
 }  
}

The UI of the activity should offer a container (can be a FrameLayout) described here by **R.id.fragment\_card\_container,** which have the appropiate width and height dimension (i.e. width = 336dp and height=210dp). The fragment component will adjust itself to the size of the container.

Using a transaction on the childFragmentManager this time, as described by the above highlighted code, we add the Card Details Fragment component to our MainFragment UI.

fragment\_main.xml

<?xml version="1.0" encoding="utf-8"?>  
<layout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 <data>  
  
 </data>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainFragment">  
  
 …  
  
 <FrameLayout  
 android:id="@+id/fragment\_card\_container"  
 android:layout\_width="336dp"  
 android:layout\_height="178dp"  
 android:layout\_marginBottom="24dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 tools:background="@color/light\_blue"  
 tools:minHeight="209dp">  
  
 </FrameLayout>  
  
 …  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
</layout>

A screenshot of a computer program

Description automatically generated with medium confidence

The MainFragment will receive a callback on completion of the card component operation described by the SuccessErrorResponse object:

override fun onCardDetailsFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(MainActivity.TAG, "MainFragment ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(MainActivity.TAG, "MainFragment ${**it**.error} ${**it**.errorMessage}")  
 **}**}

### Displaying Set/Verify/Change PIN component from an Activity

All three components Set/Verify/Change PIN are following the same pattern, so in order to integrate them we have to follow the same flow.

First step is to offer an implementation for their public interface:

SetPinFragment.OnFragmentInteractionListener,

VerifyPinFragment.OnFragmentInteractionListener or

ChangePinFragment.OnFragmentInteractionListener,

in order to receive the result of operation object which is of type SuccessErrorResponse.

Second step is to use the **specific** factory method:

SetPinFragmentFromActivity.newInstance(niInput, pinLength),

VerifyPinFragmentFromActivity.newInstance(niInput, pinLength) or

ChangePinFragmentFromActivity.newInstance(niInput, pinLength)

to get an instance of the component (ending with the “FromActivity” suffix) and call the show() method, which require an instance of the supportFragmentManager

in order to display the PIN pad on the screen.

class MainActivity : AppCompatActivity(),   
 SetPinFragment.OnFragmentInteractionListener,  
 VerifyPinFragment.OnFragmentInteractionListener,  
 ChangePinFragment.OnFragmentInteractionListener {

private lateinit var binding: ActivityMainBinding  
 private lateinit var viewModel: MainViewModel  
 private val niInput: NIInput  
 get() = makeInputObject()  
 private val pinLength: NIPinFormType  
 get() = viewModel.getPINLength()  
  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 setArchitectureComponents()  
 initializeUI()  
 setViewModelData()  
 }  
  
 private fun setArchitectureComponents() {  
 viewModel = ViewModelProvider(this)[MainViewModel::class.*java*]  
 binding = DataBindingUtil.setContentView(this, R.layout.*activity\_main*)  
 binding.*lifecycleOwner* = this  
 binding.*viewModel* = viewModel  
 }  
  
 private fun initializeUI() {  
 binding.*apply* **{** recyclerView.*apply* **{** *layoutManager* = LinearLayoutManager(*context*)  
 *adapter* = EntriesListAdapter()  
 setHasFixedSize(true)  
  
 this@MainActivity.viewModel.entriesItemsLiveData.observe(this@MainActivity) **{** itemModels **->** itemModels?.*let* **{** (*adapter* as EntriesListAdapter).setItems(**it**)  
 **}  
 }  
 }** setPinButton.setOnClickListener **{** val dialog = SetPinFragmentFromActivity.newInstance(niInput, pinLength)  
 dialog.show(*supportFragmentManager*, SetPinFragmentFromActivity.TAG)  
 **}** verifyPinButton.setOnClickListener **{** val dialog = VerifyPinFragmentFromActivity.newInstance(niInput, pinLength)  
 dialog.show(*supportFragmentManager*, VerifyPinFragmentFromActivity.TAG)  
 **}**

changePinButton.setOnClickListener **{** val dialog = ChangePinFragmentFromActivity.newInstance(niInput, pinLength)  
 dialog.show(*supportFragmentManager*, ChangePinFragmentFromActivity.TAG)  
 **}  
 }** }  
  
 private fun setViewModelData() {  
 if (viewModel.entriesItemModels.isEmpty()) {  
 val entries = *listOf*(  
 EntriesItemModel(*BANK\_CODE*, getString(R.string.*bank\_code\_txt*), "EAND"),  
 EntriesItemModel(*CARD\_ID*, getString(R.string.*card\_identifier\_id\_txt*), "52913582150735206039"),  
 EntriesItemModel(*CARD\_TYPE*, getString(R.string.*card\_identifier\_type\_txt*), "EXID"),  
 EntriesItemModel(*ROOT\_URL*, getString(R.string.*root\_url\_txt*), "https://apitest.network.ae"),  
 EntriesItemModel(*TOKEN*, getString(R.string.*token\_txt*), "ybp4s3utm9grex68r765rh2s"),  
 EntriesItemModel(*PIN\_LENGTH*, getString(R.string.*pin\_length\_txt*), NIPinFormType.*FOUR\_DIGITS*.name, getString(R.string.*pin\_length\_placeholder*))  
 )  
 viewModel.setEntriesItems(entries)  
 }  
 }private fun makeInputObject(): NIInput {  
 return NIInput(  
 bankCode = viewModel.entriesItemModels.*first* **{** model **->** model.id == *BANK\_CODE* **}**.value,  
 cardIdentifierId = viewModel.entriesItemModels.*first* **{** model **->** model.id == *CARD\_ID* **}**.value,  
 cardIdentifierType = viewModel.entriesItemModels.*first* **{** model **->** model.id == *CARD\_TYPE* **}**.value,  
 connectionProperties = NIConnectionProperties(  
 viewModel.entriesItemModels.*first* **{** model **->** model.id == *ROOT\_URL* **}**.value,  
 viewModel.entriesItemModels.*first* **{** model **->** model.id == *TOKEN* **}**.value,  
 ),  
 displayAttributes = NIDisplayAttributes(  
// fonts = listOf(  
// NIFontLabelPair(  
// UIFont(  
// R.font.architects\_daughter,  
// 18  
// ),  
// NILabels.CARD\_HOLDER\_NAME\_LABEL  
// ),  
// NIFontLabelPair(  
// UIFont(  
// R.font.architects\_daughter,  
// 16  
// ),  
// NILabels.CARD\_NUMBER\_VALUE\_LABEL  
// )  
// ),  
 cardAttributes = NICardAttributes(  
 shouldHide = false,  
 backgroundImage = R.drawable.*card\_image\_custom\_es*,  
// textPositioning = TextPositioning(  
// leftAlignment = 0.08f,  
// cardNumberGroupTopAlignment = 0.95f,  
// dateCvvGroupTopAlignment = 0.7f,  
// cardHolderNameGroupTopAlignment = 0.5f  
// )  
 ),  
// setPinMessageAttributes = PinMessageAttributes(  
// successAttributes = SuccessErrorScreenAttributes(  
// layoutId = R.layout.activity\_success,  
// buttonResId = R.id.doneButton  
// ),  
// errorAttributes = SuccessErrorScreenAttributes(  
// layoutId = R.layout.activity\_error,  
// buttonResId = R.id.doneButton  
// )  
// ),  
 verifyPinMessageAttributes = PinMessageAttributes(  
 successAttributes = SuccessErrorScreenAttributes(  
 layoutId = R.layout.*activity\_success*,  
 buttonResId = R.id.*doneButton* ),  
 errorAttributes = SuccessErrorScreenAttributes(  
 layoutId = R.layout.*activity\_error*,  
 buttonResId = R.id.*doneButton* )  
 ),  
 changePinMessageAttributes = PinMessageAttributes(  
 successAttributes = SuccessErrorScreenAttributes(  
 layoutId = R.layout.*activity\_success*,  
 buttonResId = R.id.*doneButton* ),  
 errorAttributes = SuccessErrorScreenAttributes(  
 layoutId = R.layout.*activity\_error*,  
 buttonResId = R.id.*doneButton* )  
 ),  
 theme = R.style.*Theme\_NICardManagementApp\_DayNight,*

language = NILanguage.*ARABIC* )  
 )  
 }  
  
 companion object {  
 const val TAG = "SDKLogMessage"  
 }  
  
 override fun onSetPinFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(TAG, "SetPinFragmentFromActivity ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(TAG, "SetPinFragmentFromActivity ${**it**.errorMessage}")  
 **}** }  
  
 override fun onChangePinFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(TAG, "ChangePinFragmentFromActivity ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(TAG, "ChangePinFragmentFromActivity ${**it**.errorMessage}")  
 **}** }  
  
 override fun onVerifyPinFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(TAG, "VerifyPinFragmentFromActivity ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(TAG, "VerifyPinFragmentFromActivity ${**it**.errorMessage}")  
 **}** }  
  
}

Set/Verify/Change PIN components factory method requires two paramenters: an NIInput object, and the NIPinFormType which is setting the expected number of digits for the PIN number.

enum class NIPinFormType(val minSize: Int, val maxSize: Int) {  
 *DYNAMIC*(4, 6),  
 *FOUR\_DIGITS*(4, 4),  
 *FIVE\_DIGITS*(5, 5),  
 *SIX\_DIGITS*(6, 6)  
}

pinLength = NIPinFormType.*FOUR\_DIGITS*

### Displaying Set/Verify/Change PIN component from another Fragment

In order to integrate the Set/Verify/Change PIN component from another fragment we have to follow the same steps as before, but with a few differences:

First step is to offer an implementation for their public interface:

SetPinFragment.OnFragmentInteractionListener,

VerifyPinFragment.OnFragmentInteractionListener or

ChangePinFragment.OnFragmentInteractionListener,

in order to receive the result of operation object which is of type SuccessErrorResponse.

Second step is to use the **specific** factory method:

SetPinFragmentFromFragment.newInstance(niInput, pinLength),

VerifyPinFragmentFromFragment.newInstance(niInput, pinLength) or

ChangePinFragmentFromFragment.newInstance(niInput, pinLength)

to get an instance of the component (ending with the “FromFragment” suffix) and call the show() method, which require an instance of the childFragmentManager

in order to make display the PIN pad on the screen.

class MainFragment : Fragment(),  
 CardDetailsFragmentBase.OnFragmentInteractionListener,  
 SetPinFragment.OnFragmentInteractionListener,  
 VerifyPinFragment.OnFragmentInteractionListener,  
 ChangePinFragment.OnFragmentInteractionListener {  
  
 private var \_binding: FragmentMainBinding? = null  
 private val binding: FragmentMainBinding  
 get() = \_binding!!  
  
 private lateinit var niInput: NIInput  
 private lateinit var niPinFormType: NIPinFormType  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *arguments*?.*getSerializableCompat*<NIInput>(Extra.EXTRA\_NI\_INPUT)?.*let* **{** niInput = **it  
 }** ?: throw RuntimeException("${this::class.*java*.*simpleName*} arguments serializable ${Extra.EXTRA\_NI\_INPUT} is missing")  
  
 *arguments*?.*getSerializableCompat*<NIPinFormType>(Extra.EXTRA\_NI\_PIN\_FORM\_TYPE)?.*let* **{** niPinFormType = **it  
 }** ?: throw RuntimeException("${this::class.*java*.*simpleName*} intent serializable ${Extra.EXTRA\_NI\_PIN\_FORM\_TYPE} is missing")  
 }  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?): View? {  
 // Inflate the layout for this fragment  
 \_binding = DataBindingUtil.inflate(inflater, R.layout.*fragment\_main*, container, false)  
 return binding.*root* }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)binding.setPinButton.setOnClickListener **{** val dialog = SetPinFragmentFromFragment.newInstance(niInput, niPinFormType)  
 dialog.show(*childFragmentManager*, SetPinFragmentFromFragment.TAG)  
 **}** binding.verifyPinButton.setOnClickListener **{** val dialog = VerifyPinFragmentFromFragment.newInstance(niInput, niPinFormType)  
 dialog.show(*childFragmentManager*, VerifyPinFragmentFromFragment.TAG)  
 **}** binding.changePinButton.setOnClickListener **{** val dialog = ChangePinFragmentFromFragment.newInstance(niInput, niPinFormType)  
 dialog.show(*childFragmentManager*, ChangePinFragmentFromFragment.TAG)  
 **}** }  
  
 override fun onDestroyView() {  
 super.onDestroyView()  
 \_binding = null  
 }  
  
 override fun onSetPinFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(MainActivity.TAG, "SetPinFragmentFromFragment ${**it**.message}") **}** response.isError?.*let* **{** Log.d(MainActivity.TAG, "SetPinFragmentFromFragment ${**it**.error} ${**it**.errorMessage}") **}** }  
  
 override fun onVerifyPinFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(MainActivity.TAG, "VerifyPinFragmentFromFragment ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(MainActivity.TAG, "VerifyPinFragmentFromFragment ${**it**.error} ${**it**.errorMessage}")  
 **}** }  
  
 override fun onChangePinFragmentCompletion(response: SuccessErrorResponse) {  
 response.isSuccess?.*let* **{** Log.d(MainActivity.TAG, "ChangePinFragmentFromFragment ${**it**.message}")  
 **}** response.isError?.*let* **{** Log.d(MainActivity.TAG, "ChangePinFragmentFromFragment ${**it**.error} ${**it**.errorMessage}")  
 **}** }  
  
 companion object {  
 @JvmStatic  
 fun newInstance(input: NIInput, type: NIPinFormType) = MainFragment().*apply* **{** *arguments* = Bundle().*apply* **{** putSerializable(Extra.EXTRA\_NI\_INPUT, input)  
 putSerializable(Extra.EXTRA\_NI\_PIN\_FORM\_TYPE, type)  
 **}  
 }** const val TAG = "MainFragment"  
 }  
}

enum class NIPinFormType(val minSize: Int, val maxSize: Int) {  
 *DYNAMIC*(4, 6),  
 *FOUR\_DIGITS*(4, 4),  
 *FIVE\_DIGITS*(5, 5),  
 *SIX\_DIGITS*(6, 6)  
}

pinLength = NIPinFormType.*FOUR\_DIGITS*

The NIPinFormType enum represent the valid number of digits for the PIN operations and is described by the following members:

- *DYNAMIC* : between 4 and 6 digits inclusive (i.e. PIN with 4, 5 or 6 digits are accepted as valid)

- *FOUR\_DIGITS* : a fixed number of 4 digits

- *FIVE\_DIGITS* : a fixed number of 5 digits

- *SIX\_DIGITS* : a fixed number of 6 digits

This is a frontend parameter which is used to enable the “Done” button on the Set/Verify/Change PIN screens when the certain number of digits are reached as a result of using the input pad.

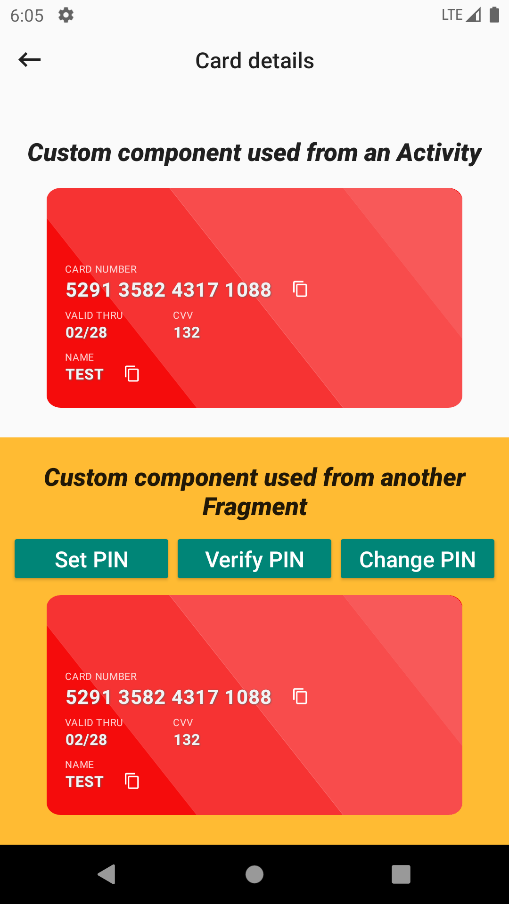
# The SDK User Interface

## Display Card Details UI

The Card Details Form API is opening a new Activity on the screen with a Card Details Fragment in the center of the screen and a custom navigation component on the top.



The Card Details Fragment component can be integrated into an activity or a fragment and represent, actually, the card UI on the screen:



The NICardAttributes class is described by the following parameters:

- shouldHide if true, the card details will be hidden/masked by default; if false, the card details will be visible by default

- backgroundImage if set, this image will be used as background for the card details view; if not set, it will use default image from sdk

- textPositioning if set will apply new text positioning values for Card Details components: cardNumberLine, dateCvvLine and cardHolderNameLine, if not set default positioning will be used

We are exposing some of the scenarios related to parameters for a more clear understanding of how to use NICardAttributes:

* shouldHide = true

|  |  |  |
| --- | --- | --- |
|  |  |  |

The *card* *number* and the *card* *holder* *name* have two corresponding buttons with a meaning of “copy to clipboard’ which can be use to copy and share the information.

* shouldHide = false

|  |
| --- |
|  |

Setting the shouldHide parameter to false result in displaying the card component without the “eye” pictogram and displaying by default all the card details to the user.

* backgroundImage = R.drawable.default\_upi

Setting the backgroundImage to a new value will replace the default background image (left) resource of the card with the new one (right):

|  |  |
| --- | --- |
|  |  |

* textPositioning = TextPositioning(

leftAlignment = 0.08f,

cardNumberGroupTopAlignment = 0.95f,

dateCvvGroupTopAlignment = 0.7f,

cardHolderNameGroupTopAlignment = 0.5f

)

Setting the textPositioning will override the default values (left) with the new one (right):

|  |  |
| --- | --- |
|  |  |

Default position have the following equivalent values:

textPositioning = TextPositioning(

leftAlignment = 0.04f,

cardNumberGroupTopAlignment = 0.52f,

dateCvvGroupTopAlignment = 0.7f,

cardHolderNameGroupTopAlignment = 0.89f

)

All parameters values are interpreted as a percent of the parent container width/height i.e. :

- leftAlignment = 0.04f will be interpreted as 4% of the parent container width

- cardNumberGroupTopAlignment = 0.52f will be interpreted as 52% of the parent container height with a reference point of left|top corner (axis origin point).

A screenshot of a computer

Description automatically generated with medium confidence

## Set PIN Component UI

Set PIN process involves two steps, enter the desired PIN value, and reenter the value for validation process.

If the second value does not match, the user will have to try again until it makes a match with the first value or have the option to navigate back to cancel the process.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

## Verify PIN Component UI

Verify PIN process is made using a single step. The user enters the value and tap the check mark done button in green.

|  |  |  |
| --- | --- | --- |
|  |  |  |

## Change PIN Component UI

Change PIN is made using one step for the current PIN value and two steps validation for the new PIN value.

If the second value does not match with the first one, the user needs to try until it makes a match, or to navigate back in order to cancel the process.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

## Localization of the SDK (Current Language Support)

The SDK has support for English (EN) and Arabic (AR) languages, with a default value for English (EN).

|  |  |
| --- | --- |
|  |  |
|  |  |

## Light / Dark mode UI

The SDK has support for Light/Dark mode.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

# Sample App

There is a sample app available at request, which can be used as a reference for integrating the NICardManagementSDK into your app.

**6. SDK Errors**

**6.1. Invalid PIN Error**

* **Error code: 889**

Occurring when there is an issue with the card PIN.

Message: Card/PIN Issue

**6.2. Specific Errors**

For now, only GENERAL\_ERROR and NETWORK\_ERROR are used!

* **GENERAL\_ERROR**

This is a general error

Message: SDK General Error

* **NETWORK\_ERROR**

This is a general error occurring when there an issue with the request to the server or no data is received.

Message: Network Error

* **NAV\_ERROR**

Occurring when there is an issue with the navigation towards a screen.

Message: Form not allowed pushing on navigation controller

* **PARSING\_ERROR**

Occurring when the data received from the server cannot not be processed.

Message: SDK Parsing Error

* **RSAKEY\_ERROR**

Occurring when a public key cannot not be generated.

Message: Couldn’t get or generate Public Key

* **PINBLOCK\_ERROR**

Occurring when there is an issue regarding the pin.

Message: PIN Block Error

* **PINBLOCK\_ENCRYPTION\_ERROR**

Occurring when there is an issue regarding the pin encryption.

Message: PIN Block Encryption Error