BlockID Android SDK (v1.7.00)

This document is for version 1.7.00

Change Log

Date (sdk_version)	**Summary**	**Action required** (for existing app/s)

Apr 18, 2022 (v1.7.00)	• Introduce new method set to dvcId (refer pages #22)
	• Document verification (refer pages #52)
	  ○ Method signature updated
	  ○ Added verification options (dl_authenticate, face_compare)
	  ○ Added sample data map
	• Error code added (refer pages #90)
	• Face compare method removed as Document verification supports face compare	

Mar 25, 2022 (v1.6.20)	• Introduce capability to register and authenticate FIDO2 security keys (refer pages #82) • Added error codes for FIDO2 capability (refer page #87)	• Refer Build.gradle dependency section (page #19)
Mar 22, 2022 (v1.6.10)	• Improvements - ○ Added additional events for analytics	

Mar 16, 2022 (v1.6.00)	• Improvements -
	  ○ Supports National ID scanning feature in one step
	• Bugs Fixed -
	  ○ Fixed ‘invalid driver license’ issue’
	  ○ Crash fixed during QR code scanning	

Feb 2, 2022 (v1.4.96)	LiveID Scanning • SDK now provides an optional method to start LiveID scan with face liveness check (page #53) • Added K_LIVENESS_CHECK_FAILED error code (page #85)	
Jan 31, 2022 (v1.4.95)	• Supports Passport scanning feature in one step	

	• Enables an option to bypass PoI (Proof of Identity) based on License module configuration	
Jan 24, 2022 (v1.4.94)	• Driver License PDF417 scanning improvement	• Refer Build.gradle dependency section (page #18)
Jan 13, 2022 (v1.4.93)	• Updated Document Verification Service section (page #50) ○ The SDK now has the capability for SSN Verification • Updated Error code section (page #82)	
Dec 20, 2021	• Added Document Verification Service (page #50) ○ The SDK now supports Driver License verification API (page #50) • The LiveIDScannerHelper class now has new constructor supporting an option to NOT to RESET LiveId scan when expression goes wrong (page #52)	
Dec 6, 2021	• Introduced an ability to set proxy for outbound API calls (page # 24)	
Oct 29, 2021	• Project level build.gradle update (page #17)	• Refer Build.gradle dependency section (page #18)
Oct 22, 2021	• Optional parameter ***isDataRequiredOnFail*** added to document (DL, Passport, National ID) scanner helper initializer method (page #28, page #34, page #41) • Added a public method named ***compareFaces()*** to allow application to compare two faces (page #59) • Updated error code section related to invalid document (page #81) ○ removed error code 412 - invalid document	

	○ new error codes added related to invalid document following format 412XXX	
Oct 11, 2021	Start RFID scanning with custom time out (page #36)	
Sep 21, 2021	Below expressions removed in LiveId scan (page #46) ● LookUp ● LookDown	
Sep 16, 2021	LiveID enhancement with below changes ● 2 emotions added (page #46) ○ LookUp ○ LookDown ● Added androidx.camera:camera-core:1.0.0-rc05 dependency in app level build.gradle (page #18)	
Sep 9, 2021	LiveID enhancement with below changes ● 4 emotions ○ Blink ○ Smile ○ LookRight ○ LookLeft ● Added new dependency in app level build.gradle (page #18) ● New overridden method expressionDidReset added in ILiveIDResponseListener (page #49)	
Aug 24, 2021	● Silent Notification ○ Driver License, Passport and National ID scanning process will send a broadcast named “BlockIDFaceDetectionNotification” which will provide an object with count of faces detected by scanned	

	against “numberOfFaces” (page #32, #38, #44) ○ App can register to this notification, if required any customization on UI ● registerDocument (page #30, #36, #42, #49) ○ New method added to register document which has additional parameter storeArtifact and syncArtifact	
Aug 18, 2021	● Added commitRestorationData() and resetRestorationData() method in restore (page #64) ● Error code changed for K_TENANT_REGISTRATION_FAIL (page #66) ● New error code introduced (page #69) ○ public key is not available (K_PUBLIC_KEY_REQUIRED) ○ encryption failed (K_ENCRYPTION) ○ decryption failed (K_DECRYPTION)	
Aug 6, 2021	● Enable Proguard in application (page #15) ● Added error code 300 (page #66)	
Jul 23, 2021	● From now SDK supports minSdkVersion = 23 (page #14) ● Removed library dependency in build.gradle (com.journeyapps:zxing-android-embedded :4.2.0) (page #14) ● Updated library dependency in build.gradle com.google.android.gms:play-services-visi on:20.1.3 (page #14) ● DLScannerHelper constructor argument changed, now it required BIDSacnnerView (page #25) ● New method added for start and stop driver license scanning in DLScannerHelper (page #26) ● Added overridden 2 new methods in IDriverLicenseResponseListener (page #26)	

Jul 20, 2021	• Removed Firebase Analytics dependency from build.gradle (page #14)	
Jul 16, 2021	• BlockID SDK package name has been changed to **com.onekosmos.blockid.sdk** • Added new library dependency in build.gradle (page #14) • Changes related to LiveId - mandatory for identity documents and not for miscellaneous documents • Error message updated for K_LIVEID_CANNOT_BE_ENROLLED (page #67)	
Jul 8, 2021	• Changed signature of authenticateUser() method; userId argument can now be Nullable or any valid String value	
Jul 1, 2021	• userId argument in authenticateUser() is Optional • authenticateUser() method callback argument updated, sessionId argument can be Null now, the method returns sessionId • New method introduced in IBiometricResponseListener	
Jun 21, 2021	• Set minSdkVersion to 24 • Added new library dependency in build.gradle (com.journeyapps:zxing-android-embedded:4.2.0)	
Jun 17, 2021	• Added Table of Contents • Document formatting and minor spelling corrections	
Jun 14, 2021	• Identity document mandatory parameters camelcase are changed (firstName and lastName)	
Jun 10, 2021	• Added Table of Contents • Contents formatted and restructured	
Jun 7, 2021	• Classes such as BIDDocumnetData, BIDDriverLiecense, BIDPassport and BIDNationalID are deprecated.	

	LinkedHashMap is used. The relevant methods below are changed • BIDDocumnetType enum has been removed from the register documents method changes can be found on the save document section of each document type • Change in License Checks / Modules • New section in the SDK functions tile is added to get the server public key • Removed typo errors	
May 21, 2021	• Added dependency in application level Build.gradle • Enroll liveId response object and setLiveId method argument change • Set pin method argument change • Unenrollment document method argument change • New error code added	
May 20, 2021	• Added a new method in the user consent section for pre-set data • Changes in Document enrollment methods and added new notes before trying to save the data • Alternate method of saving documents is added • Verify liveid method is updated • The Get Document with filter method is added in the document enrollment section	
May 11, 2021	• Added dependency in Application level Build.gradle for RFID scan	
May 7, 2021	New method added • Initiate wallet • Commit wallet	
May 4, 2021	Response parameters added in Enroll Driver License	
Apr 30, 2021	• Updated biometric gradle dependency. • Added a new argument in the enrollDeviceAuth(), unEnrollDeviceAuth(), verifyDeviceAuth() methods	
Apr 22, 2021	Released first draft of this document	

Table of Contents

Overview

This document describes the procedure to configure the BlockID Android SDK into your application. This integration will allow your users to use the features provided within the Android SDK of the BlockID mobile application. The features include enrollment of the user’s LiveID (captures the live (real-time) facial images) details, Fingerprint, PIN, Driving License (DL), Passport details and National Id. Also, It has various types of authentication mechanisms by which a user can log in to its system to access it.

Dependencies

You will need the following resource to complete this integration

- Android Studio Bumblebee

Configurations that need to be performed to enable this integration

- Configure BlockID Android SDK within the project

BlockID SDK Setup

The setup consists of four factors

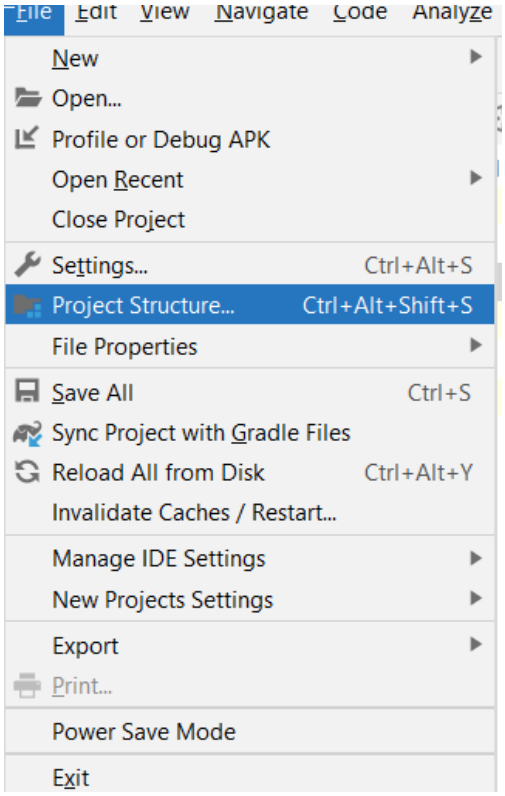
1. Integrating the SDK module into your project
2. Java Compatibility
3. Enabling Permissions
4. Add SDK dependencies in the gradle file

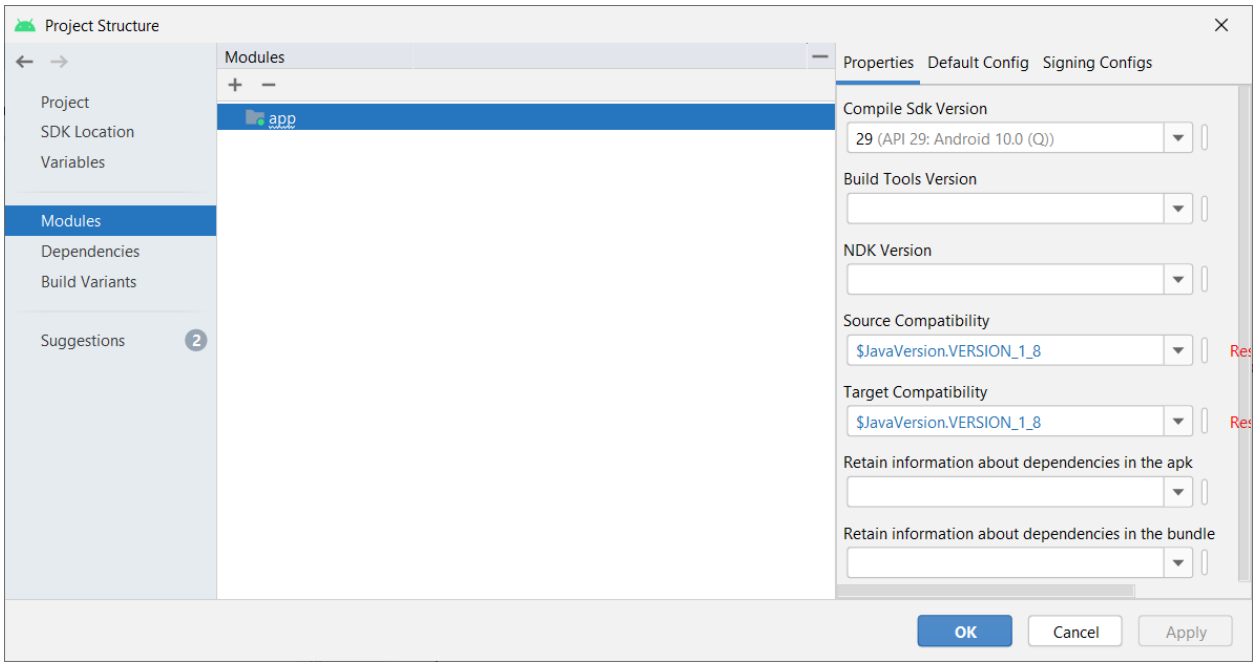
Add BlockID SDK .aar file

Perform the following steps to integrate the BlockIDSDK_{version}-Release.aar file into your app's project.

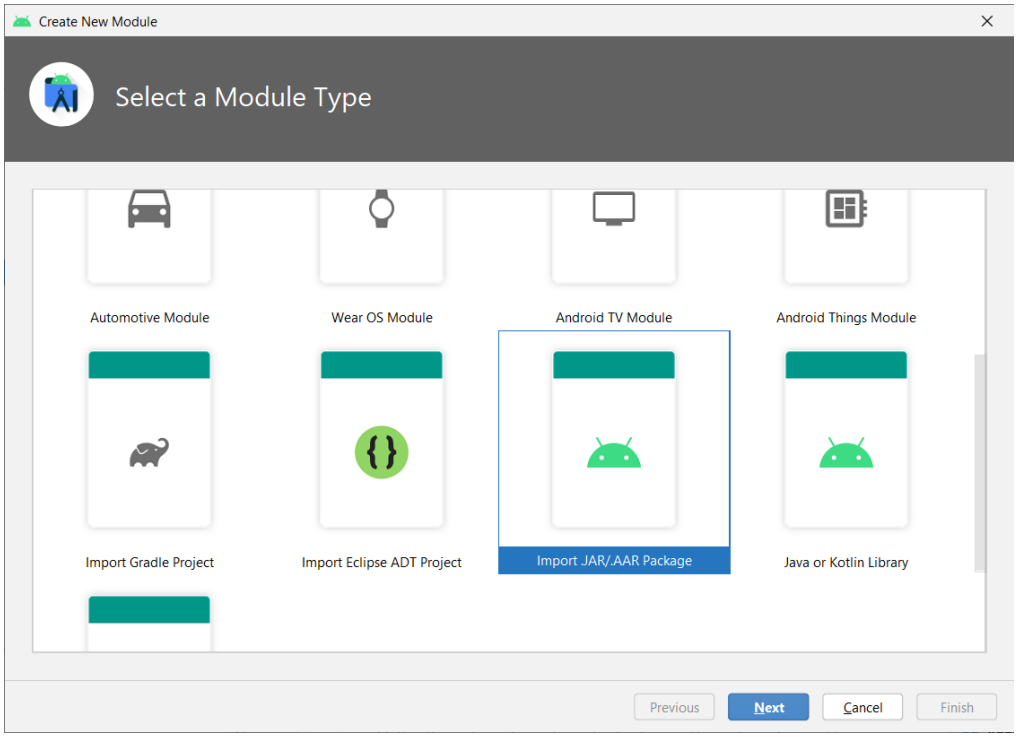
NOTE: while adding the aar file into the app please be mindful to rename the file by removing the version number from the file.

1. Start the Android Studio
2. Navigate to File > Project Structure > Modules

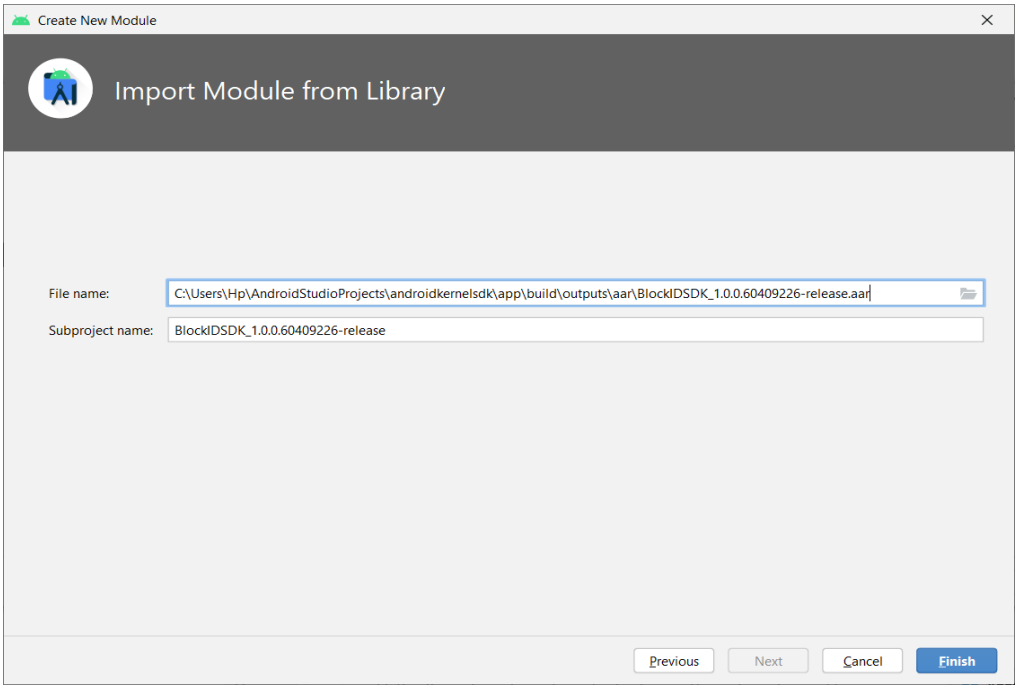

File Edit View Navigate Code Analyze
New
Open...
Profile or Debug APK
Open Recent
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
File Properties
Save All Ctrl+S
Sync Project with Gradle Files
Reload All from Disk Ctrl+Alt+Y
Invalidate Caches / Restart...
Manage IDE Settings
New Projects Settings
Export
Print...
Power Save Mode
Exit


Project Structure
Project
SDK Location
Variables
Modules
Dependencies
Build Variants
Suggestions 2
Modules
app
Properties Default Config Signing Configs
Compile Sdk Version
29 (API 29: Android 10.0 (Q))
Build Tools Version
NDK Version
Source Compatibility
$JavaVersion.VERSION_1_8
Target Compatibility
$JavaVersion.VERSION_1_8
Retain information about dependencies in the apk
Retain information about dependencies in the bundle
OK
Cancel
Apply

3. Click on the + icon
4. Click choose import .JAR/.AAR

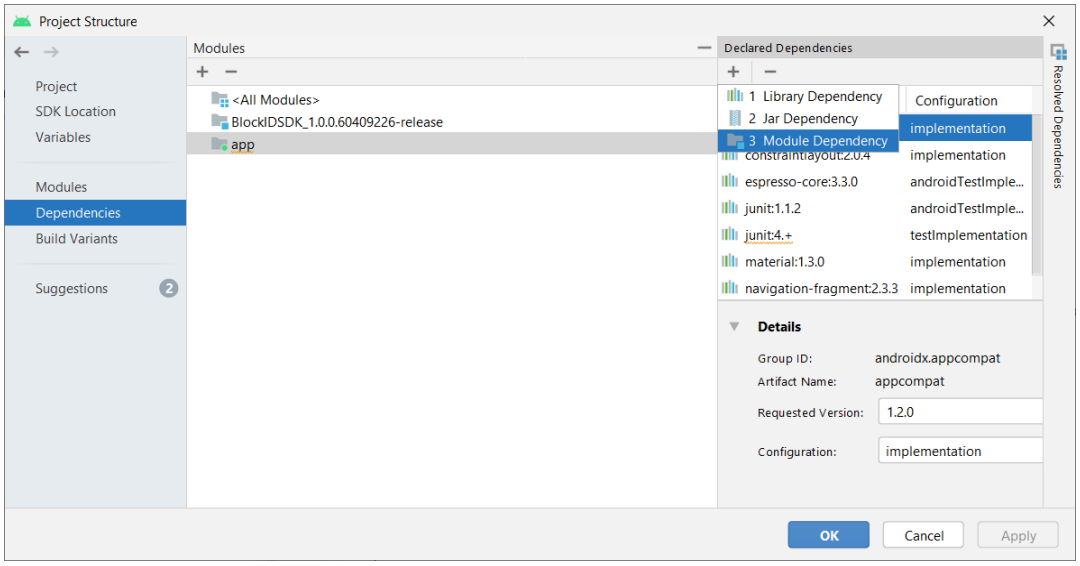


5. Locate the BlockIDSDK_{version}-Release.aar file and click appl

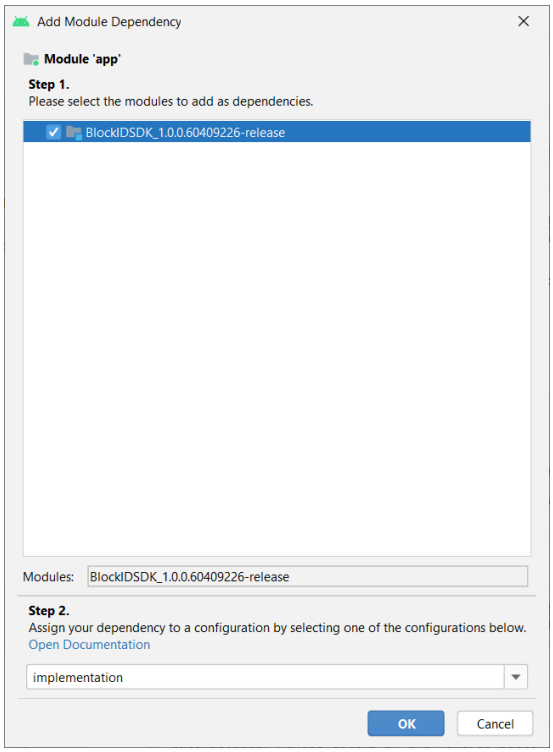


6. For Module dependency

Navigate to File > Project Structure > Dependencies > app > declared dependencies tab > Module dependenc



7. Select BlockIDSDK_{version}-Release



8. Click apply then click OK. The build process for the project starts

SDK dependency in project

Use the following code to add BlockIDSDK_{version}-Release.aar dependency into your project's **Build.gradle file**

Use the following code to add BlockIDSDK-Release.aar dependency into your project's Build.gradle file

```
        implementation project(path: ':BlockIDSDK-release');
```

Java Compatibility

The SDK source and target compatibility is now set to Java 8. The SDK is no longer binary compatible with applications that target Java 7. In order to use this and future releases, developers must upgrade their applications to target Java 8. Follow the snippet below for reference

```
android {
        compileOptions {
                sourceCompatibility 1.8
                targetCompatibility 1.8
        }
}
```

Update Android Manifest

In order to target Android API level 23 or later, you will need to ensure that your application requests runtime permissions for camera, internet access.

Add the following to your Android Manifest file in <application> tag

```
tools:replace="android:allowBackup,android:label,android:theme"
```

Add the following to your Android Manifest file

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Request permissions in your application code

```
ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.{your_permission},
PERMISSION_REQUEST_CODE);}
```

Build.gradle dependencies

NOTE: SDK supports minSdkVersion 23

Project level Build.gradle

```
allprojects {
   repositories {
      google()
      jcenter()
      mavenCentral()
      maven { url "https://jitpack.io" }
      maven { url 'http://www.baka.sk/maven2' }
   }
}
```

Application level Build.gradle

Dependency	Purpose
com.google.code.gson:gson:2.8.5	Gson
androidx.security:security-crypto:1.0.0-rc04	Secure Shared preference
androidx.lifecycle:lifecycle-extensions:2.2.0	lifecycle-extensions
androidx.lifecycle:lifecycle-runtime:2.2.0	lifecycle-extensions
com.google.android.gms:play-services-vision:20.1.3	Qr Code scan and Driver License scanning
org.web3j:core:3.3.1-android	wallet

implementation('org.bitcoinj:bitcoinj-core:0.15.4') {
exclude group: 'com.google.protobuf', module: 'protobuf-java'
}
	generate mnemonics

androidx.biometric:biometric:1.1.0	fingerprint and biometric
org.jetbrains:annotations:16.0.1	fingerprint and biometric
commons-codec:commons-codec:1.10	Base32 encode/decode
com.amitshekhar.android:android-networking:1.0.2	Network Call
implementation('org.jmrtd:jmrtd:0.7.19') {	passport scan

exclude module: 'bcprov-jdk15on' exclude module: 'bctls-jdk15on' }	
org.innovatrics.mrz:mrz-java:0.4	passport scan
androidx.multidex:multidex:2.0.1	RFID Scan
net.sf.scuba:scuba-sc-android:0.0.20	RFID Scan
com.madgag.spongycastle:prov:1.54.0.0	RFID Scan
com.github.mhshams:jnbis:1.1.0	RFID Scan
com.gemalto.jp2:jp2-android:1.0	RFID Scan
com.annimon:stream:1.2.2	Filter document
androidx.camera:camera-camera2:1.1.0-alpha12	CameraX
androidx.camera:camera-lifecycle:1.1.0-alpha12	
androidx.camera:camera-view:1.0.0-alpha32	
com.google.android.gms:play-services-mlkit-face-detection:16.1.7	LiveID
com.google.guava:guava:27.1-android	
com.google.android.gms:play-services-mlkit-text-recognition:17.0.0	Text recognition
com.google.mlkit:barcode-scanning:17.0.0	Barcode detector
com.quickbirdstudios:opencv:4.5.2	Image processing
androidx.browser:browser:1.4.0	Custom tab intent

Enable Proguard

Add below rules in proguard-rules.pro file, if the app code requires to be obfuscated.

```
-keep class com.onekosmos.blockid.sdk.** {  *; }
-keep public class org.** {  *; }
```

SDK Initialization

The SDK is now added to your application, in order to get the features of the SDK, you need to add the license key and tenant details. You have to set the following parameters in your project.

NOTE: these must be done in the same sequence as listed below

1. Initialize SDK
2. Set License key
3. Create wallet
4. Register tenant
5. Commit wallet

NOTE: We recommend using the setup code in the MainApplication class of your application. "set license" function should be called on every launch of the application. Any time the license key is changed (from previous launches), the SDK will self-reset and all stored data will be lost.

Other functions (create, register, commit) should be called **only** if the SDK is not ready. See method isReady() below.

Initialize SDK

To initialize the sdk just use the code below

```
BlockIDSDK.initialize(Context);
```

The above code will initialize the SDK key components. Now the SDK is initialized, one needs to set the license key so that the SDK can check whether you are authorized to use the SDK or not.

Set License key

To set the license Key use the below code.

```
BlockIDSDK.getInstance().setLicenseKey(licenseKey);
```

Request Parameters

Params	Description
licenseKey	String value of license key

After setting the license key the SDK will automatically check whether the license key is valid or not and will set it in the SDK so that one can access the features of SDK.After setting the license key one should call the tenant registration to complete the sdk.

Set dvcId(Document verification connecter id)

To set the dvcId use the below code.

```
BlockIDSDK.getInstance().setDvcId(dvcId);
```

Request Parameters

Params	Description
dvcId	String

Initialize wallet

Below function is used to initialize a Wallet.

```
BlockIDSDK.getInstance().initiateWallet();
```

Register tenant

Tenants can be represented as the value or an object on which one's API calls can be registered. There are two types of tenants one will be the root and another will be the client tenant.

Root tenant will be used to store the details regarding the user enrolled biometric and digital assets where else the user's enrolled persona will be stored on the client tenant.
Application is expected to call ‘registerTenant’ once. If an existing tenant is set, the application will load it after validating licenseKey. SDK will only allow tenant registry if and only if the license key is already added.

NOTE: A root tenant can always be a client tenant but a client tenant cannot be a root tenant
One can easily access the SDK class called BIDTenant, which one will require while tenant registration from application.To construct the BIDTenant object use following code.

```
BIDTenant defaultTenant = new BIDTenant ("Your_TAG", "Your_COMMUNITY" , "Your_DNS");
```

Params	Description
Your_TAG	String value of Tenant tag
Your_COMMUNITY	String value of community.
Your_DNS	String value of url

Now once the BIDTenant object is created then the same object can be sent for the registration by calling following method

```
BlockIDSDK.getInstance().registerTenant(BidTenant,BIDTenantRegistryCallback);
```

Request Parameters

Params	Description
BidTenant	BIDTenant object
BIDTenantRegistryCallback	Type of interface to get a response.

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String
tenant	Type of BIDTenant

Commit wallet

Below function is used to commit a temporary wallet.This function will get called on the success status of registerTenant.

```
BlockIDSDK.getInstance().commitApplicationWallet();
```

Check if the SDK is Ready

If the application license key, wallet and tenant are set. The below method will return true. Application should check this on every launch. If the SDK is not ready, all other functions will fail and may lead to a crash. If the SDK is not ready, the application is responsible for re-initialization (see Initialize Wallet, register tenant, commit wallet.

```
BlockIDSDK.getInstance().isReady()
```

Reset SDK

Call the following function if you need to reset the SDK for any reason (eg: “user enters wrong pin 5 times”).

This function purges the SDK wallet, public private key set and any other data stored / owned by the SDK.

```
BlockIDSDK.getInstance().resetSDK()
```

Get SDK Version

To get the SDK's current version, the below method should be called. The method returns the current version of the SDK in String format. (e.g. 1.4.5.xxxxxx where xxxxxx is build number generated randomly)

```
BlockIDSDK.getInstance().getVersion();
```

Enable Proxy

BlockID SDK supports adding proxy to all the outbound API calls. To support this feature, applications can use the below methods.

Set Proxy

```
BlockIDSDK.getInstance().setProxy(host, port, userName, password);
```

Params	Description
host	Non null String value
port	int value
username	Nullable String value
password	Nullable String value

NOTE: This method **must** be called before calling setLicenseKey method

Get proxy details

```
BlockIDSDK.getInstance().getProxyDetails();
```


1KOSMOS

SDK / Data Security

- To prevent any kind of unauthorized access to the SDK, there is a locking mechanism implemented in the BlockID SDK.
- The SDK will always be in the locked state.To unlock the SDK one has to successfully login/register in any mode. After that, the SDK will be unlocked and all the APIs will be accessible.
- If the SDK is locked and someone tries to access the API, it will show the message as “Unauthorized access”.
- For security purposes, on the app side, the developers have to lock the SDK again, when the app goes in the background or the user is directed to the Login screen.
- Check SDK lock status before calling any enrollment APIs of SDK for security purposes.

The following methods are exposed to the application from the SDK.

Feature	**Function / Method**
Lock SDK	BIDAuthProvider.getInstance().lockSDK()
Check if SDK is locked	BIDAuthProvider.getInstance().isSdkLocked()
Unlock SDK	BIDAuthProvider.getInstance().unlockSDK()

SDK Functions

Get Server public key

To get the server public key, the below method should be called.The method returns the value of the server public key in String format.

```
BlockIDSDK.getInstance().getServerPublicKey();
```

Get Mnemonic Phrases

Mnemonic phrases (recovery phrases) are 12 in count. These phrases are used for the restoration of a wallet in future. To get the mnemonic phrases call below function

NOTE: You only need to call this if you need to support specific UX for the user to view / backup mnemonic phrases.

```
BlockIDSDK.getInstance().getMnemonic();
```

Get Distributed Identifier(DID)

Distributed Identifier (DID) is an identifier which is globally unique. When you create a wallet, it is created along with it. All assets like documents, biometric will be enrolled against this DID.

```
BlockIDSDK.getInstance().getDID();
```

Document Enrollment

BlockID SDK allows two main categories of documents: Identity or Misc (Miscellaneous)

NOTE: Before enrolling documents, please ensure that your specific licenseKey has been authorized on the admin console. You can contact the support team for this.

NOTE: The Data Models class such as BIDDocumnetData, BIDDriverLiecense, BIDPassport and BIDNationalID are being deprecated; instead the scanner will return a single HashMap which will have the document data.

NOTE: All documents must have following baseline attributes

Attribute	Description
id	String (It represents the document's ID eg: driver license number)
category	String (Currently this SDK supports two categories for documents that can be registered i.e, identity_document or misc_document)
type	String (This represents the type of document that the user will try to register eg: pin, dl, ppt, liveid, nationalid)
proofedBy	String (This identifies the entity responsible for proofing the document. When you use BlockID SDK Scanners, this defaults to “blockid”)

NOTE: BlockID platform will restrict enrollments to approved proofers. Please contact 1Kosmos support to ensure that your entity names are approved before using another proofer.

Identity Documents must have baseline attributes

- firstName: String
- lastName: String
- dob: String <yyyyMMdd>
- doe: String <yyyyMMdd>
- face: Base64 string for the face photo
- image: Base64 string for the image of the document (eg: dl front image)

NOTE:

- for documents that carry front and back images, we recommend providing the back image as a Base64 string as well
- The date params are read from the document and then converted into yyyyMMdd format for further use

Misc. Documents do not have any additional baseline requirements.

BlockID SDK now expects a LinkedHashMap to register and unregister documents.

NOTE: Before registering an Identity document please make sure that LiveId is enrolled.
If LiveId is not already enrolled, the SDK will throw an error “LiveId is mandatory”.

NOTE: Please check for camera permissions before using any methods to scan a document.

Driver License Enrollment

To scan driver licenses, the app should call the DLScannerHelper class. This scanner scans the front and back side of the driver license.

DLSannerHelper scans OCR and face from the front side of the Driver License.
DLScannerHelper scans PDF417 barcodes from the back side of the Driver License.

The **BidScannerView** is a scanner view. The Application has to add a view in the Layout file.

Add the following ui attribute to xml to use the scanning

```
<LinearLayout
  android:id="@+id/layout_view"
  android:layout_width="wrap_content"
  android:layout_height="@dimen/dimen_0dp"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent">
   <com.blockid.sdk.cameramodule.BIDScannerView
    android:id="@+id/bid_scanner_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</LinearLayout>
```

Initialize DLScannerHelper - the below constructor will return DL data ONLY after successful scan

```
mDriverLicenseScannerHelper  = new DLScannerHelper(this, mScanningOrder,
mBIDScannerView, K_DL_EXPIRY_GRACE_DAYS, iDriverLicenseResponseListener);
```

Initialize DLScannerHelper - the below constructor will return DL data in both successful and failed scan situations

```
mDriverLicenseScannerHelper  = new DLScannerHelper(this, mScanningOrder,
mBIDScannerView, K_DL_EXPIRY_GRACE_DAYS, isDataRequiredOnFail
, iDriverLicenseResponseListener);
```

Request Parameters

Params	Description
this	Not null context of the current Activity
mScanningOrder	Object of enum DLScanningOrder • FIRST_FRONT_THEN_BACK • FIRST_BACK_THEN_FRONT It can be null. By default Order = FIRST_BACK_THEN_FRONT
mBIDScannerView	Not null custom UI object of class BIDScannerView
K_DL_EXPIRY_GRACE_DAYS	Number of days (int) to allow as grace period ahead of document expiry. The SDK will throw an error if the document has already expired. If the document expires before gracePeriod, the SDK will complete the scan and return an error <advisory> as well. It is the application's responsibility to decide if to allow enrollment.
isDataRequiredOnFail	A boolean parameter with default value - false If value is true - the app can receive DL data in case of failed scan
iDriverLicenseResponseListener	Not null Interface of type IDriverLicenseResponseListener

To start Driver License scanning use below code

```
mDriverLicenseScannerHelper.startScanning();
```

To stop Driver License scanning use below code

```
mDriverLicenseScannerHelper.stopScanning();
```

When scanning is started, the application will get a response in

IDriverLicenseResponseListener.

IDriverLicenseResponseListener has 3 overridden methods.

1) Scan back side

This method will be called when driver license is ready for back side scan

```
@Override
public void scanFrontSide() {
   // put your code here
}
```

2) Scan front side

This method will be called when driver license is ready for front side scan

```
@Override
public void scanBackSide() {
   // put your code here
}
```

3) Driver License response

This method will be called when driver license scanning is completed

```
@Override
public void onDriverLicenseResponse(LinkedHashMap<String, Object> driverLicenseMap,
String signToken, ErrorManager.ErrorResponse error) {
   // put your code here
}
```

Response Parameters

Params	Description
driverLicenseMap	Object of type LinkedHashMap
signToken	String value of signatureToken
error	Object of type ErrorManager. ErrorResponse

Register Driver License

To register a driver license object, the BlockIDSDK provides the below method. By default, the method will always store (on device) and sync (to blockchain) the DL document data.

```
BlockIDSDK.getInstance().registerDocument(context, driverLicenseMap, signToken, enrollListener)
```

The SDK also provides the below method to register documents which enables application developers to control if the DL document data are required to be stored (on device) and/or synced (to blockchain)

```
BlockIDSDK.getInstance().registerDocument(context, driverLicenseMap, signToken, storeArtifact, syncArtifact, enrollListener)
```

Request Parameters

Params	Description
context	Context of the current activity
driverLicenseMap	Object of LinkedHashMap<String, Object> with pre-set data of driver license including mandatory attributes
signToken	String value of signature token received after scanning driver license
enrollListener	Object of interface type BIDDocumentProvider.IDocumentEnrollmentListener

storeArtifact	Boolean value to store DL data in local storage
syncArtifact	Boolean value to sync DL data with Eth/IPFS]

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing • code: int • message: String

Check if the Driver License Enrolled

Below method is used to check if Driver License is enrolled / registered with the BlockID

```
BlockIDSDK.getInstance().isDriversLicenseEnrolled();
```

Unenroll Driver License

To unenroll driver license, call the following method

```
BlockIDSDK.getInstance().unRegisterDocument(context, driverLicenseMap,
unenrollListener);
```

Request Parameters

Params	Description
context	Context of the current activity
driverLicenseMap	Object of LinkedHashMap<String, Object> with pre-set data of driver license including mandatory attributes
unenrollListener	Object of interface type BIDDocumentProvider.IDocumentUnEnrollmentListener

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Number of Faces Captured

BlockIDSDK will send a Broadcast that can be captured by application or by the entity using BlockIDSDK. This notification will have following configurations

Notification Name	BlockIDFaceDetectionNotification
Notification Object	● “numberOfFaces” - Int (count of faces) ● “documentType” - RegisterDocumentType ○ DL

Passport Enrollment

Scan Passport

Add the following ui attribute to xml to use the scanning

```
<LinearLayout
  android:id="@+id/layout_view"
  android:layout_width="wrap_content"
  android:layout_height="@dimen/dimen_0dp"
  android:layout_marginTop="@dimen/dimen_8dp"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent">
```

```
  <com.blockid.sdk.cameramodule.BIDScannerView
     android:id="@+id/view_bid_scanner"
     android:layout_width="match_parent"
     android:layout_height="match_parent" />
</LinearLayout>
```

One can adjust the height and the width of the scanning frame by adding this in the java file.

```
mBIDScannerView.setScannerWidthMargin(scannerOverlayMargin, overlay);
```

Request Parameters

Params	Description
mBIDScannerView	Object of class BIDScanerView
scannerOverlayMargin	Int value of margin by which the height and width of the scanning frame can be adjusted.
overlay	View object of overlay or frame, can also be null if there is no overlay.

Initialize PassportScannerHelper - the below constructor will return Passport data ONLY after successful scan

```
mPassportScannerHelper = new PassportScannerHelper(this, scanningMode,
mBIDScannerView, mScannerOverlay, K_PASSPORT_EXPIRY_GRACE_DAYS,
passportResponseListener)
```

Initialize PassportScannerHelper - the below constructor will return Passport data in both successful and failed scan situations

```
mPassportScannerHelper = new PassportScannerHelper(this, scanningMode,
mBIDScannerView, mScannerOverlay, K_PASSPORT_EXPIRY_GRACE_DAYS,
isDataRequiredOnFail, passportResponseListener)
```

Request Parameters

Params	Description
this	Context of the current activity
scanningMode	There are two types of mode recommended mode will be always SCAN_LIVE but for demo purposes one can use SCAN_DEMO
mBIDScannerView	Custom UI object of class BIDScannerView
mScannerOverlay	Custom UI object of view ScannerOverlay
K_PASSPORT_EXPIRY_ GRACE_DAYS	Number of days (int) to allow as grace period ahead of document expiry. The SDK will throw an error if the document has already expired. If the document expires before gracePeriod, the SDK will complete the scan and return an error <advisory> as well. It is the application's responsibility to decide if to allow enrollment.
isDataRequiredOnFail	A boolean parameter with default value - false If value is true - the app can receive Passport data in case of failed scan
passportResponseListener	Interface of type IPassportResponseListener

To start Passport scanning use below code

```
mPassportScanner.startPassportScanning();
```

Response Parameters

Params	Description
bidPassport	Object of type LinkedHashMap
signToken	String value of signatureToken
error	An object of ErrorResponse containing ● code: int

	• message: String

To stop scanning use passport scanner object with the following method

```
mPassportScanner.stopScanning();
```

Scan the e-passport chip

Add the following permission in the manifest file

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

Initialize PassportScannerHelper - the below constructor will return Passport data ONLY after successful scan

```
mPassportScannerHelper = new PassportScannerHelper(this,
K_PASSPORT_EXPIRY_GRACE_DAYS, passportResponseListener();
```

Initialize PassportScannerHelper - the below constructor will return Passport data in both successful and failed scan situations

```
mPassportScannerHelper = new PassportScannerHelper(this,
K_PASSPORT_EXPIRY_GRACE_DAYS, isDataRequiredOnFail,
passportResponseListener();
```

Request Parameters

Params	**Description**
this	Context of the current activity
K_PASSPORT_EXPIRY_ GRACE_DAYS	Int value of passport expiry grace days
isDataRequiredOnFail	A boolean parameter with default value - false

	If value is true - the app can receive Passport data in case of failed scan
passportResponseListener	Interface of type IPassportResponseListener

To scan the e-passport chip use the following java method:

Start RFID scanning with default timeout 15 second

```
mPassportScannerHelper.startRFIDScanning(passportMap, sigToken);
```

Start RFID scanning with custom timeout

```
mPassportScannerHelper.startRFIDScanning(passportMap, sigToken, rfidScanTimeOut);
```

Request Parameters

Params	**Description**
passportMap	Object of type LinkedHashMap
signToken	String value of signatureToken
rfidScanTimeOut	Object of type long

When you start e-passport chip scanning. You will get E-passport chip data in onNewIntent() which is an Activity Overridden method.

Call mPassportScannerHelper.onNewIntent(tag)

Params	**Description**
mPassportScannerHelper	Object of helper which is created when RFID scanning start
tag	RFID chip data detected in onNewIntent()

Response Parameters

Params	**Description**
bidPassport	Object of type LinkedHashMap

signToken	String value of signatureToken
error	An object of ErrorResponse containing • code: int • message: String

To stop e-passport chip scanning

```
mPassportScanner.stopRFIDScanning();
```

Check is Passport enrolled

Below method is used to check if passport is enrolled / registered with the BlockID

```
BlockIDSDK.getInstance().isPassportEnrolled();
```

Register passport

To register a driver license object, the BlockIDSDK provides the below method. By default, the method will always store (on device) and sync (to blockchain) the passport document data.

```
BlockIDSDK.getInstance().registerDocument(context, ppMap, signToken, enrollListener);
```

The SDK also provides the below method to register documents which enables application developers to control if the passport document data are required to be stored (on device) and/or synced (to blockchain).

```
BlockIDSDK.getInstance().registerDocument(context, ppMap, signToken, storeArtifact,
syncArtifact, enrollListener);
```

Request Parameters

Params	**Description**
context	Context of the current activity

ppMap	Object of LinkedHashMap<String, Object> with pre-set data of passport including mandatory attributes
signToken	String value of signature token received after scanning passport
enrollListener	Object of interface type BIDDocumentProvider.IDocumentEnrollmentListener
storeArtifact	Boolean value to store passport data in local storage
syncArtifact	Boolean value to sync passport data with Eth/IPFS

Response Parameters

Params	Description
status	Boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

UnEnroll passport

To unenroll your passport, call the BlockIDSDK method unregisterDocument.

```
BlockIDSDK.getInstance().unRegisterDocument(context, ppMap, unenrollListener);
```

Request Parameters

Params	Description
context	Context of the current activity
ppMap	Object of LinkedHashMap<String, Object> with pre-set data of PP including mandatory attributes
unenrollListener	Object of interface type BIDDocumentProvider.IDocumentUnEnrollmentListener

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Number of Faces Captured

BlockIDSDK will send a Broadcast that can be captured by application or by the entity using BlockIDSDK. This notification will have following configurations

Notification Name	BlockIDFaceDetectionNotification
Notification Object	● “numberOfFaces” - Int (count of faces) ● “documentType” - RegisterDocumentType ○ PPT

NationalID Enrollment

To scan NationalID, the app should call the NationalIDScanHelper class. This scanner scans the front and backside of the document.

NationalIDScanHelper scans

1. OCR and face from the front side of the document
2. OCR, MRZ and QRCode from the back side of the document

The **BidScannerView** is a scanner view. The Application has to add a view in the Layout file.

Add the following ui attribute to xml to use the scanning

```
<LinearLayout
   android:id="@+id/layout_view"
   android:layout_width="wrap_content"
   android:layout_height="@dimen/dimen_0dp"
   android:layout_marginTop="@dimen/dimen_8dp"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent">
   <com.blockid.sdk.cameramodule.BIDScannerView
      android:id="@+id/view_bid_scanner"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

One can adjust the height and the width of the scanning frame by adding this in the java file.

```
mBIDScannerView.setScannerWidthMargin(scannerOverlayMargin, overlay);
```

Request Parameters

Params	Description
mBIDScannerView	Object of class BIDScanerView
scannerOverlayMargin	Int value of margin by which the height and width of the scanning frame can be adjusted.
overlay	View object of overlay or frame, can also be null if there is no overlay

While using the methods one should determine the order in which national id will be scanned. Following are the orders in which national id can be scanned.

1. FIRST_FRONT_THEN_BACK
2. FIRST_BACK_THEN_FRONT

By default the SDK will be set to FIRST_BACK_THEN_FRONT order.

Initialize NationalIDScannerHelper - the below constructor will return NationalID data ONLY after successful scan

```
mNationalIdScannerHelper = new NationalIDScannerHelper(this,
ScanningMode.SCAN_LIVE, mScanningOrder, mBIDScannerView, mScannerOverlay,
K_NATIONAL_ID_EXPIRY_GRACE_DAYS, nationalIDResponseListener)
```

Initialize NationalIDScannerHelper - the below constructor will return NationalID both successful and failed scan situations

```
mNationalIdScannerHelper = new NationalIDScannerHelper(this,
ScanningMode.SCAN_LIVE, mScanningOrder, mBIDScannerView, mScannerOverlay,
K_NATIONAL_ID_EXPIRY_GRACE_DAYS, isDataRequiredOnFail,
nationalIDResponseListener)
```

Request Parameters

Params	Description
this	Not null context of the current activity
mScanningOrder	Object of enum NationalIDScanOrder • FIRST_FRONT_THEN_BACK • FIRST_BACK_THEN_FRONT It can be null. By default Order = FIRST_BACK_THEN_FRONT
scanningMode	There are two types of mode recommended mode will be always SCAN_LIVE but for demo purposes one can use SCAN_DEMO
mBIDScannerView	Not null custom UI object of class BIDScannerView.
mScannerOverlay	Custom UI object of view ScannerOverlay.
K_NATIONAL_ID_EXPIRY_GRACE_DAYS	Number of days (int) to allow for a grace period ahead of document expiry. The SDK will throw an error if the document has already expired. If the document expires before gracePeriod, the SDK will complete the scan and return an error <advisory> as well. It is the application's responsibility to decide if to allow enrollment.
isDataRequiredOnFail	A boolean parameter with default value - false If value is true - the app can receive National ID data in case of failed scan
nationalIDResponseListener	Not null interface of type INationalIDResponseListener

To start scanning, use national id scanner object with the following method

```
mNationalIdScannerHelper.startNationalIDScanning();
```

When scanning is started, the application will get a response in the INationalIDResponseListener.
The INationalIDResponseListener has 3 overridden methods.

1) Scan back side

This method will be called when NationalID is ready for back side scan

```
@Override
public void scanFrontSide() {
   // put your code here
}
```

2) Scan front side
This method will be called when NationalID is ready for front side scan

```
@Override
public void scanBackSide() {
   // put your code here
}
```

3) NationalID response
This method will be called when NationalID scanning is completed

```
@Override
public void onNationalIDScanResponse(LinkedHashMap<String, Object>nationalIDMap,
String signToken, ErrorManager.ErrorResponse error) {
   // put your code here
}
```

Response Parameters

Params	**Description**
nationalIDMap	Object of type LinkedHashMap
signToken	String value of signatureToken
error	An object of ErrorResponse containing ● code: int ● message: String

To stop scanning use national id scanner object with the following method

```
mNationalIdScannerHelper.stopScanning();
```

Check if NationalID enrolled
Below method is used to check if NationalID is enrolled / registered with the BlockID

```
BlockIDSDK.getInstance().isNationalIDEnrolled();
```

Register NationalID
To register a driver license object, the BlockIDSDK provides the below method. By default, the method will always store (on device) and sync (to blockchain) the nationalID document data.

```
BlockIDSDK.getInstance().registerDocument(context, nidMap, signToken, enrollListener);
```

The SDK also provides the below method to register documents which enables application developers to control if the nationalID document data are required to be stored (on device) and/or synced (to blockchain).

```
BlockIDSDK.getInstance().registerDocument(context, nidMap, signToken, storeArtifact, syncArtifact, enrollListener)
```

Request Parameters

Params	Description
context	Context of the current activity
nidMap	Object of LinkedHashMap<String, Object> with pre-set data of national id including mandatory attributes
signToken	String value of signature token received after scanning national id
enrollListener	Object of interface type BIDDocumentProvider.IDocumentEnrollmentListener
storeArtifact	Boolean value to store national id data in local storage
syncArtifact	Boolean value to sync national id data with Eth/IPFS

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Unenroll NationalID
To unenroll NationalID, call the BlockIDSDK method unregisterDocument.

```
BlockIDSDK.getInstance().unRegisterDocument(context, nidMap, unenrollListener);
```

Request Parameters

Params	Description
context	Context of the current activity
nidMap	Object of LinkedHashMap<String, Object> with pre-set data of national id including mandatory attributes
unenrollListener	Object of interface type BIDDocumentProvider.IDocumentUnEnrollmentListener

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Number of Faces Captured
BlockIDSDK will send a Broadcast that can be captured by application or by the entity using BlockIDSDK. This notification will have following configurations

Notification Name	BlockIDFaceDetectionNotification
Notification Object	● “numberOfFaces” - Int (count of faces) ● “documentType” - RegisterDocumentType ○ NATIONAL_ID

Get Enrolled Document

To get the JSONArray string of all documents use the below method.

```
BIDDocumentProvider.getInstance().getEnrolledDocumentList();
```

To get a JSONArray string of documents with filters, use the below method.

```
BIDDocumentProvider.getInstance().getUserDocument(id, type, category);
```

Request Parameters

Params	Description
id	String value of document id.
type	String value of RegisterDocType enum which has the following attributes. 1. PIN("pin") 2. LIVE_ID("liveid") 3. PPT("ppt") 4. DL("dl") 5. NATIONAL_ID("nationalid")
category	String value of category which currently has following values. 1. identity_document 2. misc_document

Note: LiveID can not get from this getUserDocument() method.

Document Verification Service

BlockID SDK provides the functionality to verify data that the user has presented through our partners.

The current version of BlockID SDK supports the below document verification services.

1. DL Authentication and Verification
2. SSN Verification
3. Check Face Liveness
4. Face Comparison

The data for the document verification can be obtained using the following means:

1. using the document scan feature of the BlockID SDK for DL Verification
2. through manual entry from the user for SSN Verification

Once the data is obtained, the following steps can be performed to verify the document.

```
// Call below function to verify document
BlockIDSDK.getInstance().verifyDocument(dvcId, documentMap, verifications, new
BlockIDSDK.IVerifyDocumentListener())
```

Request Parameters

Params	Description
dvcId	String (document verification connector id)
documentMap	LinkedHashMap<String, Object>
verifications	String[] - type of verifications, it can be one or multiple 1. DL Verification - dl_verify 2. DL Authentication - dl_authenticate 3. SSN Verification - ssn_verify 4. Check Face Liveness - face_liveness 5. Face Comparison - face_compare
IVerifyDocumentListener	BlockIDSDK.IVerifyDocumentListener interface

Response Parameters

Params	Description
status	boolean value
documentVerification	JSON string
error	An object of ErrorResponse containing • code: Int • message: String

Sample Document Map

DL Authentication

```
LinkedHashMap<String, Object> documentMap= new LinkedHashMap<String, Object>();
documentMap.put(“id”, “string_driver_license_number”); // required
documentMap.put(“type”, “string”); // required
documentMap.put(“documentType”, “DL”); // required
documentMap.put(“country”, “string”); // required
documentMap.put(“firstName”, “string”);
documentMap.put(“lastName”, “string”);
documentMap.put(“dob”, “string”);
documentMap.put(“doe”, “string”);
documentMap.put(“fullName”, “string”);
documentMap.put(“middleName”, “string”);
documentMap.put(“familyName”, “string”);
documentMap.put(“givenName”, “string”);
documentMap.put(“gender”, “string”);
documentMap.put(“height”, “string”);
documentMap.put(“eyeColor”, “string”);
documentMap.put(“street”, “string”);
documentMap.put(“city”, “string”);
documentMap.put(“state”, “string”);
documentMap.put(“zipCode”, “string”);
documentMap.put(“doi”, “string”);
documentMap.put(“discriminatorNumber”, “string”);
documentMap.put(“vehicleCodes”, “string”);
```

```
documentMap.put(“classificationCode”, “string”);
documentMap.put(“restrictionCode”, “string”);
documentMap.put(“endorsementsCode”, “string”);
documentMap.put(“inventoryControlNumber”, “string”);
documentMap.put(“residenceCity”, “string”);
documentMap.put(“residenceState”, “string”);
documentMap.put(“residenceZipCode”, “string”);
documentMap.put(“organDonor”, “string”);l
documentMap.put(“veteran”, “string”);
documentMap.put(“placeOfBirth”, “string”);
documentMap.put(“suffixName”, “string”);
documentMap.put(“nameSuffix”, “string”);
documentMap.put(“complianceType”, “string”);
documentMap.put(“socialSecurityNumber”, “string”);
documentMap.put(“akaDateOfBirth”, “string”);
documentMap.put(“akaSocialSecurityNumber”, “string”);
documentMap.put(“akaFirstName”, “string”);
documentMap.put(“akaLastName”, “string”);
documentMap.put(“akaMiddleName”, “string”);
documentMap.put(“akaGivenName”, “string”);
documentMap.put(“akaFamilyName”, “string”);
documentMap.put(“akaSuffix”, “string”);
```

DL Verification

```
LinkedHashMap<String, Object> documentMap= new LinkedHashMap<String, Object>();
documentMap.put(“id”, “string”); // required
documentMap.put(“type”, “string”); // required
documentMap.put(“back_image”, “base64_string”); // required
documentMap.put(“front_image”, “base64_string”); // required
documentMap.put(“front_image_flash”, “base64_string”);
```

SSN Verification

```
LinkedHashMap<String, Object> documentMap= new LinkedHashMap<String, Object>();
documentMap.put(“id”, “string”); // required
documentMap.put(“type”, “string”); // required
documentMap.put(“ssn”, “ssn number”); // required
documentMap.put(“firstName”, “string”);  // required
documentMap.put(“lastName”, “string”); // required
documentMap.put(“dob”, “string”);
documentMap.put(“street”, “string”);
documentMap.put(“city”, “string”);
documentMap.put(“state”, “string”);
documentMap.put(“country”, “string”);  // required
documentMap.put(“zipcode”, “string”);
```

Check Face Liveness

```
LinkedHashMap<String, Object> documentMap= new LinkedHashMap<String, Object>();
documentMap.put(“id”, “string”); // required
documentMap.put(“type”, “string”); // required
documentMap.put(“liveId”, “base64_string”); // required
```

Face Comparison

```
LinkedHashMap<String, Object> documentMap= new LinkedHashMap<String, Object>();
documentMap.put(“id”, “string”); // required
documentMap.put(“type”, “string”); // required
documentMap.put(“image1”, “base64_string”); // required
documentMap.put(“image2”, “base64_string”); // required
documentMap.put(“purpose”, “string”); //  (doc_enrollment || authentication) // FIXME
// from backend purpose is optional,
If purpose is empty, service will validate based confidence
Else validate based on purpose
```

Biometric Enrollment

LiveID Enrollment

To enroll LiveID, add the following ui attribute to xml to use the scanning

```
<LinearLayout
   android:id="@+id/layout_view"
   android:layout_width="wrap_content"
   android:layout_height="@dimen/dimen_0dp"
   android:layout_marginTop="@dimen/dimen_8dp"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent">
   <com.blockid.sdk.cameramodule.BIDScannerView
      android:id="@+id/view_bid_scanner"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

One can adjust the height and the width of the scanning frame by adding this in the java file.

```
mBIDScannerView.setScannerWidthMargin(scannerOverlayMargin, overlay);
```

Request Parameters

Params	Description
mBIDScannerView	Object of class BIDScanerView
scannerOverlayMargin	Int value of margin by which the height and width of the scanning frame can be adjusted
overlay	View object of overlay or frame, can also be null if there is no overlay

Call LiveId scanner for enrolling LiveID, create a object of scanner

- When using the below constructor, the BlockID SDK will reset LiveID scanning after the wrong expression. **The default behavior of this method will be to reset the entire LiveId scan as soon as any of the expressions goes wrong**.

```
mLiveIDScannerHelper = new LiveIDScannerHelper (this, ScanningMode.SCAN_LIVE,
liveIDResponseListener, mBIDScannerView, mScannerOverlay)
```

- The SDK provides the below constructor to override the default behavior of resetting the entire LiveId scan. If the application does not want to reset the LiveID scan, it must pass a false value to the **shouldResetOnWrongExpression** parameter.

```
mLiveIDScannerHelper  = new LiveIDScannerHelper (this, ScanningMode.SCAN_LIVE,
mBIDScannerView, mScannerOverlay, shouldResetOnWrongExpression,
liveIDResponseListener)
```

Request Parameters

Params	Description
this	Context of the current activity
scanningMode	There are two types of mode recommended mode will be always SCAN_LIVE but for demo purposes one can use SCAN_DEMO.
mBIDScannerView	Custom ui object of class BIDScannerView.
mScannerOverlay	Custom ui object of view ScannerOverlay.
shouldResetOnWrongExpression	true / false - default value is true
liveIDResponseListener	Interface of type ILiveIDResponseListener

BlockID SDK provides below 2 methods to start LiveID scanning. The second method enables a face liveness check feature.

1. To start LiveID scanning **(without face liveness check**), use below method

```
mLiveIDScannerHelper.startLiveIDScanning();
```

2. To start LiveID scanning (**with face liveness check**), use below method

```
mLiveIDScannerHelper.startLiveIDScanning(dvcID);
```

Request Parameters

Params	Description
dvcID	String (document verification connector id)

When scanning is started, the application will get a response in the ILiveIDResponseListener.

The ILiveIDResponseListener has 3 overridden methods.

1. onFaceFocusChanged

```
@Override
public void onFaceFocusChanged(boolean isFocused, String expression) {
}
```

Params	Description
isFocused	boolean type. It returns false when the face moves out of focus
expression	String value of the expression which is currently under use. ● “Blink” ● “Smile” ● “LookRight” ● “LookLeft” ● “Scanning Complete”

2. onLiveIDCaptured

```
@Override
public void onLiveIDCaptured(Bitmap liveIDBitmap, String signatureToken,
ErrorManager.ErrorResponse error) {
}
```

Params	Description
liveIdBitmap	Bitmap object of the face trying to enroll.
signatureToken	String value of signature token received after scanning liveID.

3. expressionDidReset

```
@Override
public void expressionDidReset(String message) {
}
```

Params	Description
message	Error message when expressions gets reset in case of wrong expression provided ● Reset Expression ● Face out of bounds

To Stop Scanning use the following method with a liveScanner object.

```
mLiveIDScannerHelper.stopLiveIDScanning();
```

To register LiveID use the following method

```
BlockIDSDK.getInstance().setLiveID(liveIdBitmap, liveIdProofedBy, signatureToken,
iLiveIDEnrollListener);
```

Request Parameters

Params	Description
liveIdBitmap	Bitmap object of the face trying to register
signatureToken	Type of String Send signatureToken from the value received from the listener method onLiveIDCaptured
liveIdProofedBy	String (This identifies the entity responsible for proofing the document. When you use BlockID SDK Scanners, this defaults to “blockid”)

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String ● Object: JSON String

Below SDK method is used to check if LiveID is enrolled

```
BlockIDSDK.getInstance().isLiveIDRegistered();
```

Verify LiveID: Below method is used to verify face with already enrolled LiveID

```
BlockIDSDK.getInstance().verifyLiveID(this, liveIdBitmap, listener);
```

Request Parameters

Params	Description
this	Context of current activity
liveIdBitmap	Bitmap object of the face trying to enroll.
listener	Interface of type IFaceComparisonListener

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing • code: int • message: String

Register LiveID with Document

This is the alternate method of saving the document by sending the LiveId in the method, so that the new workflow for saving the document can be achieved .Below is the method for registering the document.

```
BlockIDSDK.getInstance().registerDocument(context, documentMap, liveIdBitmap,
liveIdProofedBy, docSignToken, liveidSignToken, enrollListener);
```

The SDK also provides the below method to register documents which enables application developers to control if the document data are required to be stored (on device) and/or synced (to blockchain).

```
BlockIDSDK.getInstance().registerDocument(context, documentMap, liveIdBitmap,
liveIdProofedBy, docSignToken, liveidSignToken, storeArtifact, syncArtifact, enrollListener);
```

Request Parameters

Params	Description
context	Context of the current activity
documentMap	Object of LinkedHashMap<String, Object> with pre-set data of document including mandatory attributes
liveIdBitmap	Bitmap object of the face trying to register.
liveIdProofedBy	String (This identifies the entity responsible for proofing the document. When you use BlockID SDK Scanners, this defaults to “blockid”)
docSignToken	String value of signature token received after scanning document.
liveidSignToken	String value of signature token received after scanning liveId.
enrollListener	Object of interface type BIDDocumentProvider.IDocumentEnrollmentListener
storeArtifact	Boolean value to store live id data in local storage
syncArtifact	Boolean value to sync live id data with Eth/IPFS

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Device Auth Enrollment

Enroll Device Auth

```
BIDAuthProvider.getInstance().enrollDeviceAuth(this, title, desc, isPinFallBackRequired, biometricResponseListener);
```

Request Parameters

Params	Description
this	Context of current activity.
title	Title of the biometric dialog box
desc	Description of biometric dialog box
isPinFallBackRequired	boolean value
biometricResponseListener	Interface of type of class IBiometricResponseListener

IBiometricResponseListener has two methods onBiometricAuthResult(success, error) and onNonBiometricAutn(isKeyguardService)

onBiometricAuthResult method Response Parameters

Params	Description
success	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

onNonBiometricAutn method Response Parameters

Params	Description
isKeyguardService	Boolean value. True / false **NOTE:** This will be true when android device do not have fingerprint hardware and face lock In that case the SDK will call the keyguard server Intent. The response will get the caller activity result with response code K_KEYGUARD_RESPONSE = 4003

Check if the Device auth is enrolled.

```
BlockIDSDK.getInstance().isDeviceAuthEnrolled();
```

Unenroll Device auth.

```
BIDAuthProvider.getInstance().unEnrollDeviceAuth(this, title, desc, isPinFallBackRequired,
biometricResponseListener);
```

Request Parameters

Params	Description
this	Context of current activity
title	Title of the biometric dialog box
desc	Description of biometric dialog box
isPinFallBackRequired	boolean value True - The prompt with pin fallback False - The prompt without pin fallback
biometricResponseListener	Interface of type of class IBiometricResponseListener

Response Parameters

Params	Description
success	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Verify Device Auth

Below method is used to verify device auth with already enrolled with BlockID

```
BlockIDSDK.getInstance().verifyDeviceAuth(this, title, desc ,isPinFallBackRequired,
biometricResponseListener);
```

Request Parameters

Params	Description
this	Context of current activity.
title	Title of the biometric dialog box.
desc	Description of biometric dialog box .
isPinFallBackRequired	boolean value True - The prompt with pin fallback False - The prompt without pin fallback
biometricResponseListener	Interface of type of class IBiometricResponseListener

Response Parameters

Params	Description
success	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Pin Enrollment

```
BlockIDSDK.getInstance().enrollPin(pin, proofedBy, listener);
```

Request Parameters

Params	Description
pin	String value of application pin
proofedBy	String value of pro
listener	Interface of type BIDDocumentProvider.IDocumentEnrollmentListener

Response Parameters

Params	Description
success	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Check if the pin is enrolled.

```
BlockIDSDK.getInstance().isPinRegistered();
```

Unenroll pin

```
BlockIDSDK.getInstance().unenrollPin(listener);
```

Request Parameters

Params	Description
listener	Interface of type BIDDocumentProvider.IDocumentUnEnrollmentListener

Response Parameters

Params	Description
success	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Verify Pin

Below method is used to verify the pin with an already enrolled pin with BlockID. The method returns Boolean value.

```
BlockIDSDK.getInstance().verifyPin(pin);
```

Request Parameters

Params	Description
pin	String value of application pin

License Checks / Modules

License checks are necessary for getting the state of modules, whether they are enabled for the license. Below methods returns the Modules enabled for the specific enrollments

To get Biometric Enrollments

```
Enrollments.BiometricAssets enrollments =
BlockIDSDK.getInstance().getBiometricAssetEnrollments();
```

To Check Specific Enrollment one can use it in these way

```
enrollments.EnrollBiometric
enrollments.EnrollLiveID //It is used for registering the LiveID.
enrollments.EnrollPin
enrollments.ScanLiveID // It is used for scanning the LiveID.
```

To get Digital Assets Enrollment

```
Enrollments.DigitalAssetsenrollments =
BlockIDSDK.getInstance().getDigitalAssetEnrollments();
```

To Check Specific Enrollment one can use it in these way

```
enrollments.ScanDL
enrollments.ScanPP
enrollments.ScanNationalID
enrollments.EnrollIdentityDocument
enrollments.EnrollMiscDocument
```

QR Scan

Add the following ui attribute to xml to use the scanning

```
<LinearLayout
   android:id="@+id/layout_view"
   android:layout_width="wrap_content"
   android:layout_height="@dimen/dimen_0dp"
   android:layout_marginTop="@dimen/dimen_8dp"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent">
   <com.blockid.sdk.cameramodule.BIDScannerView
      android:id="@+id/view_bid_scanner"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

One can adjust the height and the width of the scanning frame by adding this in the java file.

```
mBIDScannerView.setScannerWidthMargin(scannerOverlayMargin, overlay);
```

Request Parameters

Params	Description
mBIDScannerView	Object of class BIDScanerView
scannerOverlayMargin	Int value of margin by which the height and width of the scanning frame can be adjusted.
overlay	View object of overlay or frame, can also be null if there is no overlay.

To scan QR for login within the system or website and for adding or linking an account , you must have access to the QRScannerHelper. This class provides the QRScanner, which returns the QRScanned data.

```
new QRScannerHelper(this, ScanningMode.SCAN_LIVE, listener,
mBIDScannerView).startQRScanning();
```

Request Parameters

Params	Description
this	Context of the current activity
scanningMode	There are two types of mode recommended mode will be always SCAN_LIVE but for demo purposes one can use SCAN_DEMO
mBIDScannerView	Custom ui object of class BIDScannerView
mScannerOverlay	Custom ui object of view ScannerOverlay
Listener	Interface of type IOnQRScanResponseListener

Response Parameters

Params	Description
qrData	String value of QR Scan results.

To Stop the QR Scanning called the qrScannerHelper following method

```
QRScannerHelper.stopQRScanning();
```

User Linking

Magic link (get info, redeem)

When a magic link is resolved to an app, the app must Call GET AccessCode API. This will return info on what authType is required for the access code.

a. IF it's otp, app should use pre-existing method of opening the ACR web component

b. IF it's authn, then the app must present a native UI asking the user for userName/password

To Call GET AccessCode API

```
BlockIDSDK.getInstance().checkIfADRequired(code, tag, api, community, adValidationCallback);
```

Request Parameters

Params	Description
code	String value of code received from the magic link.
tag	String value of tag received from the magic link.
api	String value of the url is received from the magic link.
community	String value of the community received from the magic link.
adValidationCallback	Interface of type IADValidationCallback

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int

	• message: String
bidGenericResponse	Response object of type BIDGenericResponse
userId	String value of userId

To call Redeem

```
BlockIDSDK.getInstance().verifyUserAccount(code, userId,  password, api, tag, community, BuildConfig.VERSION_NAME, mLatitude, mLongitude, pushId, addScepAccountCallback);
```

Request Parameters

Params	Description
code	String value of code received from the magic link.
userId	String value of userId received from the user.
password	String value of password received from the user.
api	Value of ‘api’ key from magic link b64 decoded data
tag	Value of ‘tag’ key from magic link b64 decoded data
community	Value of ‘community’ key from magic link b64 decoded data
BuildConfig.VERSION_NAME	Current version name of the app
mLatitude	latitude of current location
mLongitude	longitude of current location
pushId	Push notification ID received from the firebase while installing the app.
addScepAccountCallback	Interface of type AddScepAccountCallback

Response Parameters

Params	Description

status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

Authentication

There are three types in which users can be linked.

1. By Account link through web url.
2. By Adding a scep account .
3. By Adding a Non-scep account.

By Account link through web url.

After scanning the qr code the qr code payload will consist of the model called BIDOrigin.BIDOrigin will then have a method of **getAuthPage();** by calling this method one will get the authpage authpage cannot be null if it's null then need to show an appropriate error for it. if the auth page contains url the app should open the webview once the webview is open the app should show a dialog box of 6 digit alphanumeric code if the code matches with the opened webview url the user should click yes it's a match button on the dialog box and should enters the userid and password of the account which needs to be linked. After authentication the webview the webview will request the encrypted payload after decrypting the payload one will receive the userId then the user id should be linked by calling the following method.

```
BlockIDSDK.getInstance().linkUser(userId, pushid, bidOrigin, callback);
```

Request Parameters

Params	Description
userId	String value of userId received from qr scanning.
bidOrigin	BIDOrigin Object constructed from the data received from the qr scanning.
pushid	Firebase push id for sending the push notification.
callback	Interface of type of LinkUserCallback

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String

2. By Adding a scep account

If the auth page after scanning contains the native auth schema like "blockid://authenticate" and the method is scep (which means userid should have a valid signature token and hash to be able to login in the windows.) When the method is equal to scep then one should ask the user for its AD creds(username and password) after getting the username and password from the user one should call the following api to validate the user and link the user id.

```
BlockIDSDK.getInstance().addNativeAccount(pushId, userId, password), bidOrigin,
BuildConfig.VERSION_NAME, mLatitude, mLongitude, addScepAccountCallback);
```

Request Parameters

Params	Description
userId	String value of userId received from qr scanning.
password	String value of user password.
bidOrigin	BIDOrigin Object constructed from the data received from the qr scanning.
pushid	Firebase push id for sending the push notification.
BuildConfig.VERSION_NAME	Current app version name
mLatitude	Latitude of current location
mLongitude	Longitude of current location
addScepAccountCallback	Interface of type AddScepAccountCallback

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing • code: int • message: String

3. By Adding a Non-scep account.

The whole process will be the same as above the only difference will be the method value will be null or something else then it will be considered as a non scep user which will be linked but signature token and hash for windows login will not be present.

Authentication (UWL workflows)

User Consent

The user consent screen represents the QRscopes asked while scanning qr code. The qrCodeData string received after scanning QR code isa base64 encoded string. This has to be converted to a JSON object after decoding. Following code snippet shows the QRScopes to be present on the UserConsent screen.and to get the data of the scopes the following method can be used.

```
BlockIDSDK.getInstance().getScopes(userId, scopes, creds, origin);
```

Request Parameters

Params	Description
userId	String value of userId received from qr scanning.
scopes	String value of scopes (comma separated string)received from qr scanning .
creds	String value of creds received from qr scanning.
origin	BIDOrigin Object constructed from the data received from the qr scanning.

This method will return an object type of BIDGenericResponse from SDK.

After getting the scope data from the SDK as a user one will be asked for the authentication to verify that the user itself is going to login on the consent screen after the successful authentication api should be called to call the same following method can be used.

For authentication without pre-set data use below method

```
BlockIDSDK.getInstance().authenticateUser(userId, sessionId, scopes, creds, origin, lat, lon, BuildConfig.VERSION_NAME, authenticateUserCallback);
```

For authentication with pre-set data use below method

```
BlockIDSDK.getInstance().authenticateUser(userId, sessionId, dataObject, creds, origin, lat, lon, BuildConfig.VERSION_NAME, authenticateUserCallback);
```

Request Parameters

Params	Description
userId	Nullable or valid String value
sessionId	String value of sessionID received from qr scanning **NOTE:** sesionId can be null / empty string
scopes	String value of scopes (comma separated string)received from qr scanning.
dataObject	Object of LinkedHashMap<String, Object> which has preset Data hashMap.
creds	String value of creds received from qr scanning.
origin	BIDOrigin Object constructed from the data received from the qr scanning.
lat	Latitude of the current location
lon	Longitude of the current location
BuildConfig.VERSION_NAME	Current version name of the app
authenticateUserCallback	Interface of type of AuthenticateUserCallback

Response Parameters

Params	Description
status	boolean value
sessionId	String
error	An object of ErrorResponse containing • code: int • message: String

Offline Authentication

Offline authentication means, get the user logged into your windows, when your phone is in offline mode.

For making this feature available, you should first have to link a user with DID and the user should be logged in once with windows in online mode.

To get the count of linked users with the application, use the below method.

```
BlockIDSDK.getInstance().getLinkedUserList();
```

This method will return an object type of BIDGenericResponse from SDK. To check if the offline is supported to the current user

```
BlockIDSDK.getInstance().getSelectedAccount().getDataObject();
```

The above method will give the current user info selected by the app user.

```
BlockIDSDK.getInstance().getSelectedAccount().getDataObject().ifOfflineAuthSupported(selectedAccount);
```

The above method will give the info on if it supports offline authentication.Were **selectedAccount** is an object received from the previous method.

To Generate QRCode for authentication following method can be used

```
BlockIDSDK.getInstance().getOfflineAuthPayload(selectedAccount);
```

This method will return an object type of BIDGenericResponse from SDK.Were **selectedAccount** is an object received from the above method.To create the time interval packets from the received BIDGenricResponse object use the object following with **getDataObject().toString().**

Details:-

if the selected persona does not have a associated workstationauthorigin entry, then the clicking on offline on home tab will show message (error)

Unable to login offline with a selected account.

When user attempts to access offline auth on device, app will

- identify the origin based on the persona selected
- find the WorkstationAuthOrigin that has the same origin.
- For generating offline QR payload
    - use the WorkstationAuthOrigin.scopes to prepare the QR payload similar to how we would if this was a QR scan.
    - Use the selected persona to find the account details (eg: userId, scep hash, token that needs to be sent: and any other info per scopes in line above)
    - use the WorkstationAuthOrigin.publicKey for ECDSA.
- If user cancels the offline (click cancel, tap home tab, application in background, switch to another tab), Exit the offline auth workflow

Restore

If a user wants to restore his wallet on another device, User needs to provide 12 mnemonic phrases which the user has saved while registering. These phrases will be used to get the wallet back and restore all documents.

To get the wallet using mnemonic phrases, use the below method.

```
BlockIDSDK.getInstance().restoreWallet(mnemonicPhrasesList);
```

Request Parameters

Params	Description
mnemonicPhrasesList	Object of type List of strings of 12 mnemonic phrases.

To Set restore mode on. You have to call this method before restoration begins.

```
BlockIDSDK.getInstance().setRestoreMode();
```

To set the tenant details after generating wallet and setting restore mode on. This function sets the tenant details.

```
BlockIDSDK.getInstance().registerTenant(bidTenant, callback);
```

Request Parameters

Params	Description
bidTenant	Type of BIDTenant
callback	Interface type of BIDTenantRegistryCallback.

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String
tenant	Type of BIDTenant

To Fetch the digital assets for successful restoration Use below methods

```
BlockIDSDK.getInstance().restoreUserDataFromWallet(listener);
```

Request Parameters

Params	Description
listener	Interface type of IAccountRestoreListener.

Response Parameters

Params	Description
status	boolean value
error	An object of ErrorResponse containing ● code: int ● message: String
message	String value of message

To commit restored wallet use below code

```
BlockIDSDK.getInstance().commitRestorationData();
```

To reset restore wallet data use below code

```
BlockIDSDK.getInstance().resetRestorationData();
```

Security Check

BlockID SDK provides three types of security checks for the application.

Check Device Auth

Use the below method to get the authentication type and if this authentication type is enabled on a device.

```
BlockIDSDK.getInstance().checkDeviceFingerPrintEnabled();
```

It returns the object of the BIDGenericResponse class of the SDK.

Check Device Security

```
BlockIDSDK.getInstance().checkIfDeviceIsSecured();
```

Or

```
BlockIDSDK.getInstance().isDeviceSecure(getApplicationContext());
```

It returns the object of the BIDGenericResponse class of the SDK.

FIDO2 Capability

The below methods in the BlockID SDK enable the capability to register and authenticate FIDO2 security keys.

Register Key

The application must call the below method to perform registration of the FIDO2 security key.

```
BlockIDSDK.getInstance().registerFIDOKey(activity, userName, tenantDNS, communityName, registerFIDOKeyCallback)
```

Request Parameters

Params	Description
activity	Activity: Not Null - the application's activity on top of which the registration screen will be shown
userName	String; Not Null
tenantDNS	String; Not Null
communityName	String: Not Null
RegisterFIDOKeyCallback	RegisterFIDOKeyCallback: Not Null

Response Parameters

Params	Description
status	boolean
error	An object of ErrorResponse containing ● code: int ● message: String

Authenticate Key

The application must call the below method to perform authentication of the FIDO2 security key.

BlockIDSDK.getInstance().authenticateFIDOKey(activity, userName, tenantDNS, communityName, authenticateFIDOKeyCallback)

Request Parameters

Params	Description
activity	Activity: Not Null - the application's activity on top of which the registration screen will be shown
userName	String; Not Null
tenantDNS	String; Not Null
communityName	String: Not Null
AuthenticateFIDOKeyCallback	AuthenticateFIDOKeyCallback: Not Null

Response Parameters

Params	Description
status	boolean
error	An object of ErrorResponse containing • code: int • message: String

Error Codes

This section provides a list of all error codes used in this guide.

Error Keys	Code	Message
K_CONNECTION_ERROR	0	No internet connection.
K_TENANT_REGISTRATION_FAIL	1001	Tenant registration fail
K_LIVE_ID_NOT_MATCH	401	LiveID did not match
K_PP_DL_NOT_MATCH	410	Passport and Driver license data don't match.
K_LIVEID_DOC_FACE_NOT_MATCH	411	Your photo on the document does not match with LiveID enrolled on this device.
K_DL_BACK_FRONT_DATA_NOT_MATCH	413	Driver License Front and back don't match.
K_DOCUMENT_ABOUT_TO_EXPIRE	414	This document is about to expire.
K_DOCUMENT_ALREADY_EXPIRED	415	The document you are trying to enroll in has already expired.
K_DOCUMENT_ENROLLED	416	Document is already enrolled
K_PP_NID_NOT_MATCH	417	Passport and National ID data don't match.
K_DL_NID_NOT_MATCH	418	Driver License and National ID data don't match.
K_DL_SCAN_CANCEL	419	Cancel Driver License scanning
K_LIVEID_IS_MANDATORY	421	LiveID is mandatory
K_ID_IS_MANDATORY	422	Document ID is mandatory

K_TYPE_IS_MANDATORY	423	Document type is mandatory
K_CATEGORY_IS_MANDATORY	424	Document category is mandatory
K_PROOFED_IS_MANDATORY	425	Document proofed by is mandatory
K_LIVE_ID_NOT_ENROLLED	426	LiveID can't be un-enrolled
K_INVALID_CATEGORY	427	Document category is invalid
K_DOCUMENT_NOT_EXISTS	428	Document does not exist
K_FIRSTNAME_IS_MANDATORY	429	Document firstname is mandatory
K_LASTNAME_IS_MANDATORY	430	Document lastname is mandatory
K_DOB_IS_MANDATORY	431	Document date of birth is mandatory.
K_DOE_IS_MANDATORY	432	Document date of expiry is mandatory.
K_FACE_IS_MANDATORY	433	Document face is mandatory
K_IMAGE_IS_MANDATORY	434	Document image is mandatory
K_LIVEID_ENROLLED	435	LiveID is already enrolled
K_DOCUMENT_DATA_MIS_MATCH	436	Your document data doesn't match with enrolled document
K_LIVEID_CANNOT_BE_ENROLLED	437	LiveID can only be enrolled using setLiveID and registerDocument(with LiveID) methods

K_SSN_IS_MANDATORY	438	SSN is mandatory
K_PP_ABOUT_TO_EXPIRE	100001	This document is about to expire.
K_PP_ALREADY_EXPIRED	100002	Expired passport.
K_INVALID_PP_BIO_DATA	100004	Passport Bio-Data not scanned properly
K_INVALID_PP_E_CHIP_DATA	100005	Invalid E-Passport Chip Data
K_PP_NOT_MATCH_WITH_DL	100006	Passport and Driver license data does not match.
K_PP_NOT_MATCH_WITH_DL	100006	Passport and Driver license data does not match.
K_PP_NOT_MATCH_WITH_NID	100007	Passport and National ID data does not match.
K_PP_FACE_NOT_MATCH_WITH_PROFILE	100008	Your passport photo does not match with your profile photo.
K_PP_RFID_TIMEOUT	100009	E-Passport Chip scan timeout.
K_OFFLINE_AUTH_UNSUPPORTED	510	Unable to login offline with a selected account.
K_RESTORE_DOCS_FAILED	610	Account restoration failed.
K_RESTORE_LIVE_ID_FAILED	611	Your LiveID verification failed.
K_NO_MIGRATION_NEEDED	418	No migration needed
K_USERID_ALREADY_EXITS	1005	UserId already exists
K_SOMETHING_WENT_WRONG	1007	Something went wrong!\nPlease try again!

K_UNAUTHORIZED_ACCESS	1111	Unauthorized access
K_LICENSE_KEY_NOT_ENABLED	1112	License key is not enabled for this module
K_PUBLIC_KEY_REQUIRED	1200	Public key required
K_ENCRYPTION	1201	Something went wrong with encrypting information
K_DECRYPTION	1202	Something went wrong with decrypting information
K_DOCUMENT_DATA_MANDATORY	412000	Document data is mandatory
K_INVALID_DL	412001	Invalid Driver License
K_INVALID_PP	412002	Invalid Passport
K_INVALID_NID	412003	Invalid National ID
K_DOCUMENT_VERIFICATION_FAILED	412004	Scanned document verification failed
K_INVALID_BASE64_IMAGE	412005	Invalid base64 image
K_LIVENESS_CHECK_FAILED	412006	LiveID Liveness check failed
FIDO2 Capability		
K_NULL_ACTIVITY	413001	Activity object is mandatory
K_USERNAME_IS_MANDATORY	413002	Username is mandatory
K_TENANT_DNS_IS_MANDATORY	413003	Tenant DNS is mandatory

K_COMMUNITY_NAME_IS_MANDATORY	413004	Community name is mandatory
K_CANCELED_PROCESS	413005	The process is terminated by user
K_SESSION_EXPIRED	413006	Session has expired
Document Verification		
K_SERVICE_URL_NOT_SET	414001	Document verification service URL is not set for environment