Lumidigm® M21x Android SDK

Design Guide

PLT-04606, Rev. A.0

September 2019

hidglobal.com

Revision history

Date	Description	Revision
September 2019	Initial release.	A.0

Contacts

For additional offices around the world, see **www.hidglobal.com/contact/corporate-offices**.

Americas and Corporate	**Asia Pacific**
611 Center Ridge Drive Austin, TX 78753 USA Phone: +1 866 607 7339 Fax: +1 949 732 2120	19/F 625 King's Road North Point, Island East Hong Kong Phone: +852 3160 9833 Fax: +852 3160 4809
Europe, Middle East and Africa (EMEA)	**Brazil**
Haverhill Business Park, Phoenix Road Haverhill, Suffolk, CB9 7AE United Kingdom Phone: +44 (0) 1440 711 822 Fax: +44 (0) 1440 714 840	Condomínio Business Center Av. Ermano Marchetti, 1435 Galpão A2 - CEP 05038-001 Lapa - São Paulo / SP Brazil Phone: +55 11 5514-7100

HID Global Technical Support: **www.hidglobal.com/support**.

Contents

1 Introduction 5
1.1 Components 5
1.2 Platform support and limitations 5
1.3 Getting started 5
2 Basic SDK functionality 6
2.1 Initializing the device 6
2.2 Synchronous and asynchronous modes 7
2.2.1 Asynchronous mode 7
2.2.2 Synchronous mode 7
2.3 Capturing a fingerprint image, template and PAD result 7
2.4 Verifying a biometric template 8
2.5 Retrieving version information 8
2.6 Retrieving device configuration 8
2.7 Setting device configuration 8
2.8 Error handling 8
2.9 Setting security levels 9
2.10 Best practice 9
2.11 Troubleshooting 10
2.12 Error codes 10

This page is intentionally left blank.

1 Introduction

This document describes how to communicate with the Lumidigm® M21x series of sensors, using the HID Global® M21x Android SDK Revision 1.0. For further information, refer to:

- *Lumidigm M21x Android SDK Quick Start Guide* (PLT-04563) which provides a quick guide to setting up Android Studio and building the supplied sample code.
- *Lumidigm M21x Android SDK Source Code Example* (PLT-04564).

1.1 Components

The only required component for using the SDK is the biosdk-x.y.z.aar library. All required resources and assets are located inside the AAR library. For more information on the AAR library format, go to **https://developer.android.com/studio/projects/android-library**. This AAR exposes a Java interface.

1.2 Platform support and limitations

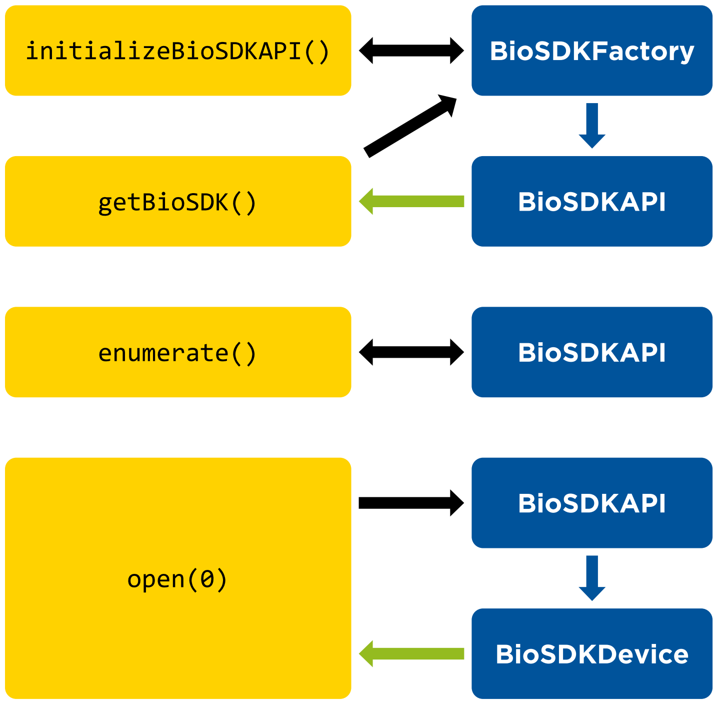
The Android SDK release 0.2 supports Android OS versions 5.0 - 9.0, hardware platforms ARMv7, ARMv8, and X86_64. This currently covers an estimated 90% of deployed Android hardware/software platform solutions. Any application that is authored using this SDK will have the same set of hardware and software limitations.

1.3 Getting started

It is strongly recommended that you start by building the sample code and validating that the sample application can be deployed on the target platform. Once the sample application is validated, you are ready to start writing your own application code.

2 Basic SDK functionality

2.1 Initializing the device



The client must request and initialize the BioSDK from the BioSDKFactory. On success (BIOSDK_OK) use the `BioSDKFactory.getBioSDK()` call to retrieve a BioSDKAPI interface. This interface can be used to request that any applicable connected USB devices enumerate, using the call `BioSDKAPIObject.enumerate`(…). Device enumeration includes asking the user for run-time permissions to use the USB device, and sending status messages back to the client asynchronously via the BioSDKVisitor listener passed into the call to `enumerate`(…) by the client. See the sample code and SDK API documentation for more information on these methods and interfaces. If the user of the client application fails to grant permission, the client application receives notification of this failure and must deal with this event appropriately, as the USB device cannot be accessed until the permissions are granted. See the sample application code for a reference implementation which can be altered for your needs.

If the enumeration is successful, you can now open a device using `BioSDKAPIObj.open(0)`. Note that the parameter passed in is ignored, as it is intended for future compatibility. Upon success, the `open` function will return a BioSDKDevice interface, which represents the primary object you will use to communicate with the sensor.

2.2 Synchronous and asynchronous modes

You must decide whether to communicate with the device synchronously or asynchronously. The API exposes both modes of operation.

2.2.1 Asynchronous mode

Asynchronous API calls are labeled via the schema `memberfunc_asynch`. Asynchronous API calls are designed to be called from your main GUI thread, and require you to implement the appropriate listener based on the call that is being made. For example, `capture_asynch` requires the implementation of `ICaptureListener`.

Callbacks to the implemented listener functions may also be assumed to be on the GUI thread, with the exception of the `IStatusListener`. The `IStatusListener` currently implements one call,

```
boolean onUpdateStatus(int nstatus);
```

which is used to update the status during the capture/verification, etc. through the `nstatus` field. Returning false from this function will cancel any asynchronous or synchronous capture which is currently in progress. See the sample code for a reference implementation. Note that `onUpdateStatus` is called from the main worker thread(s) in the SDK and must be allowed to unwind quickly. If you would like to post a status update to your GUI, you must do so using the various synchronization methods provided by Android. See the sample code for a reference implementation.

2.2.2 Synchronous mode

Synchronous API calls are labeled via the schema `memberfunc`. Synchronous API calls are designed to be invoked from a worker thread you create. The worker thread must handle all communication back to the main UI thread, as required.

2.3 Capturing a fingerprint image, template and PAD result

```
BioDeviceStatus capture(int timeoutSeconds, IStatusListener listener);
```

```
BioDeviceStatus capture_async(int nTimeoutSeconds, ICaptureListener listener);
```

Both the asynchronous and synchronous versions of the API calls operate as follows. When the call is initiated, all cached biometric data is cleared. The call proceeds, and if successful the image, template, PAD (presentation attack detection) result, and match result are temporarily cached in the SDK. For asynchronous calls, these values are also returned in the listeners, for convenience. After a successful transaction, you can retrieve the results of the last transaction using the calls:

```
Bitmap getLastImage();
byte[] getLastTemplate();
int getLastPADResult();
```

To clear the cached values from memory (for organizational purposes or privacy considerations) you can invoke:

```
void clearBioData();
```

See the SDK API documentation for more information.

2.4 Verifying a biometric template

Use either of the following calls to verify a captured biometric fingerprint template:

```
BioDeviceStatus verify(int timeoutSeconds, List<byte[]> templateList, IStatusListener
listener);
```

```
BioDeviceStatus verify_async(List<byte[]> templateList, int nTimeoutSeconds,
IVerifyListener listener);
```

Note: The field `templateList` may contain up to three biometric templates of the same finger. The verification rules are as follows:

Number of templates passed in	Verification criteria
1	Template must match.
2	Either template must match.
3	Two of three templates must match.

To retrieve the result, use:

```
int getLastMatchresult();
```

The return value of 1 indicates a match, 0 indicates no match.

```
int getLastPADResult();
```

The return value of 1 indicates a real finger, 0 indicates a spoof/presence attack.

See the SDK API documentation for more details on these calls.

2.5 Retrieving version information

To retrieve the version information of the SDK, spoof model, etc. use:

```
Map<String, String> getVersionInfo();
```

See the SDK API documentation for more information.

2.6 Retrieving device configuration

To retrieve the configuration state of the device, use:

```
Map<String, String> getConfigurationState();
```

See the SDK API documentation for more information.

2.7 Setting device configuration

To set the device configuration, use:

```
BioDeviceStatus setConfigurationState(Map<String, String> obj);
```

See the SDK API documentation for possible properties to set. See *Section 2.9 Setting security levels* for more information on setting the security levels of both PAD and fingerprint matching.

2.8 Error handling

See *Section 2.12 Error codes* for possible return values.

2.9 Setting security levels

Biometric performance must be tuned to fit an application's needs, requiring a trade-off between system security and user convenience.

Setting the performance of the system towards system security may increase the number of "False Rejects" (FRR), meaning a user enrolled in the system is falsely rejected when they place the correct finger for verification.

Setting the performance of the system towards convenience may increase the number of "False Accepts" (FAR), meaning an imposter has a greater chance of accessing the system with a fingerprint that does not match the enrolled fingerprint. Note that even on the "convenience" setting, the chances of this occurring are quite low. Bias towards this setting is appropriate for applications when user convenience is primary, and security is a secondary concern.

The defaults for both PAD and Matching are set to "Medium". Possible parameters for security levels are:

Key: **matching_security_level**

Value	False Accept Rate (%)	False Reject Rate (%)
"CONVENIENCE"	0.80	0.8
"MEDIUM"	0.10	1.0
"MEDIUM_HIGH"	0.05	1.4
"HIGH"	0.01	2.0

Key: **pad_security_level**

Value	False Reject Rate (%)
"CONVENIENCE"	0.5
"MEDIUM"	1
"HIGH"	5

2.10 Best practice

Enrolling multiple instances of the same finger, and passing them in for verification, greatly increases the user experience while maintaining the high security level. The trade-off is directing the user to place their finger on the sensor multiple times during enrollment, instead of just a single placement. The result is a lower False Reject Rate for verification. The business logic for this must be implemented by the application developer. It is best to make certain all templates used for enrollment match each other, at a "CONVENIENCE" matching threshold setting.

2.11 Troubleshooting

Q: After building and deploying the application, it reports that the device is not found.

A: Make certain that your platform supports USB OTG functionality. This can be checked using third-party applications found on the Google Play store. It is also important to check the quality of your cables/adapters, as low cost products can often cause issues.

Q: After initiating a capture, the sensor immediately starts then stops capturing.

A: Make certain that the onUpdateStatus interface implementation returns true as a default. Android Studio defaults the return value to false when it auto-implements the listener.

2.12 Error codes

Status code	Description
`BIOSDK_OK`	Expected result.
`BIOSDK_TIMEOUT`	User-set timeout occurred.
`BIOSDK_CANCELLED`	Reported when client code cancels biometric operation either through the cancel_async API or returning false from the listener API.
`BIOSDK_ERROR_ALREADY_INITIALIZED`	Device already initialized. May be ignored.
`BIOSDK_ERROR_NOT_INITIALIZED`	Reported when you attempt to make a biometric call without initializing the device, or when the device has not enumerated.
`BIOSDK_ERROR_NO_DATA`	Reserved for future use.
`BIOSDK_ERROR_PARAMETER`	Reported when a bad parameter is passed into the API call.
`BIOSDK_ERROR_THREAD`	Reserved for future use.
`BIOSDK_ERROR_PROCESSING`	Internal error occurred processing biometric data.
`BIOSDK_ERROR_ASYNC_TASK_RUNNING`	Reported if client attempts to call the asynchronous or synchronous API while one is already in process.
`BIOSDK_ERROR_INTERNAL`	Internal error. Close and reinitialize the SDK.
`BIOSDK_ERROR_USER_DENIED_PERMISSIONS`	Reported during an enumeration attempt if you deny the application the permissions for the USB device.
`BIOSDK_ERROR_SYSTEM_PERMISSIONS`	Reported if another process has access to the biometric device, or if the device has not been closed properly before reopening.
`BIOSDK_ERROR_NO_DEVICE_PRESENT`	Reported if the Android OS is not able to find the expected fingerprint sensor.
`BIOSDK_ERROR_ENROLLMENTS_DO_NOT_MATCH`	Reserved for future use.
`BIOSDK_FINGER_PRESENT`	Reserved for future use.

This page is intentionally left blank.

hidglobal.com