Payment Device SDK for iOS/ Android

API for Release 5.4.2 (Cassiopeia)

Table of contents

1. How to Use This Guide 9
- Caution 9
- Key Point or Important Concept 9
- Helpful Hint 9
- Reference Material 9
- Computer Text 9
- References and Hyperlinks 10

2. Document Scope 11

3. What You Will Need 12
- Payment Gateway Account 12
- Application Identifier 12
- Payment Device SDK for iOS/Android Software 12
  - Android Flavours 13
    - Lite 13
    - Full 13
  - IOS Flavours 14
    - Lite 14
    - Full 14
- Tap To Mobile Supplement 14

4. Transaction Flow 15
- Transaction Diagram 15
  - Mandatory 16
  - Optional or Configuration Dependent 17
- Transaction Result 17
- Transaction State 18

5. Configuration 19
- POS Registration 19
- Configuration Updates 19

6. Modes Of Operation 22
- Configuration 22
  - Tap To Mobile-Only Environment 22
  - Tap To Mobile and Payment Device Environment 22
  - Default Payment Device Behavior 22
- Transaction processing 23
  - Accepted Parameter Values 23

7. Configuration and Utility Methods 24
Parameters 24
Initialization 24
SetProperties 24
ConnectAndConfigure 25
Firmware Updates 25
Tap To Mobile 25
Smart POS Admin Settings 26
GetAvailablePinPads 26
GetStatus 27
Smart POS Admin Settings Status 27
Identifiers 27
GetTransactionInformation 28
GetMerchantData 28
RequestTmsUpdate 28
Disconnect 29
Dispose 29
8. Configuration and Utility Events 30
AvailablePinPads Callback 30
ConnectAndConfigureFinished Callback 30
ConfigurationUpdate Callback 31
DeviceUpdate Callback 32
TmsUpdate Callback 32
FirmwareUpdate Callback 33
Delayed Firmware Updates 33
Firmware Update Progress 33
9. Payment Methods 35
StartTransaction 35
Transaction POI - Tap To Mobile 35
Tipping 36
Card Entry Timeout 37
Auto Confirm 37
PAN Key Entry 38
Account Verification 38
Delay Online Processing 38
Surcharging 38
One way Card Tokens 40
To Request Card Tokens: 40

Card Tokens on Smart POS 40

Obtaining Feature Tokens: 40

Customer Vault 41

Credential on File 41

Incremental Authorization 42

ContinueSignatureVerification 42

ContinueSignatureCapture 43

ContinueSurcharge 43

ConfirmTransaction 44

VoidTransaction 44

LinkedRefundTransaction 45

TerminateTransaction 45

ProcessReceipt 45

10. Payment Events 47

TransactionUpdate Callback 47

CardDetails Callback 47

Request Activity Callback 48

TransactionFinished Callback 49

ProcessReceiptFinished Callback 50

SignatureVerification Callback 50

SignatureCapture Callback 51

Surcharge Callback 52

ApplicationSelection Callback 52

UserNotification Callback 53

11. Platform Specific Requirements 54

SQLCipher 54

ID TECH VP3350 SDK 54

BBPOS SDK 55

iOS 55

Swift 56

Android 57

Obfuscation 58

12. Tap To Mobile 60

iOS 60

Additional Permission Requirements 60

Updating Info.plist 60

Permission usage descriptions 61

Example 61

Apple Sandbox Account 61
Entitlements 61
Capabilities 62
Android 63
Additional Requirements 63
Additional Permissions 64
Updating Dependencies 64
Attestation Requirements 65
Display Requirements 65
13. Device Information 66
ID TECH VP3350 BLE 66
Power Button Operation 66
LED Indicators 67
BBPOS Chipper 3X BT 68
Power Button Operations 68
Power On 68
Power Off 68
Clear Bluetooth Pairing Keys 69
LED Indicator 70
Datecs Bluepad 50 71
Device Menu 71
Pairing Mode 71
Miura 71
Power On 71
Power Off 71
Device Menu 71
Enter pairing mode 71
Wifi Configuration Menu 72
Ingenico Axium 72
Wifi Configuration 72
14. Connection Requirements 73
USB / Lightning 73
ID TECH VP3350 Android 73
Supported Devices 73
ID TECH VP3350 IOS (Lightning and USB-C) 73
Supported Devices 73
Bluetooth 74
Miura 74

BBPOS Chipper 2X BT 74
BBPOS Chipper 3X BT 75
ID TECH VP3350 Android 75
ID TECH VP3350 IOS 75
Ingenico Axium 75
Wi-Fi 76
Appendix 1 - Supported PIN Pads 77
Appendix 2 - TMS Properties 78
Appendix 3 - Receipting 79
Appendix 4 - App Submission 80
Before Submission 80
IOS App Store Submission 80
Google Play Store Submission 80
Export Compliance 81
Appendix 5 - Firewall Configuration 82
Appendix 6 - Smart POS Supported API Methods 83
Appendix 7 - Smart POS Supported Features 85
Appendix 8 - Axium Device Modes 86
Development 86
Production 87

Document History

Document Version	Software Version	Date (yyyy-mm-dd)	Summary of Changes
4.0	Release 4.0.0 (Starfish)	2025-01-31	Tap To Mobile support added.
4.1	Release 4.0.1 (Starfish)	2025-03-14	Updated document version.Updated info.plist requirements for Tap To Mobile.
4.2	Release 4.0.2 (Starfish)	2025-04-17	Updated document version.
4.3	Release 4.0.3 (Starfish)	2025-05-02	Updated document version.Updated iOS Release Contents Table.
4.4	Release 4.0.4 (Starfish)	2025-05-23	Updated document version. Payment device SDK now supplied as a framework as well as a static library in the release.
4.5	Release 4.0.3 (Starfish)	2025-07-18	Updated document version.Added Android Tap To Mobile Support
4.6	Release 4.0.5 (Starfish)	2025-09-12	Updated the required permissions for Tap To Mobile
4.7	Release 4.0.6 (Starfish)	2025-11-07	Updated document version. Added Xcode minimum version information.
4.8	Release 4.0.6 (Starfish)	2025-11-28	Updated document version.Corrected SQLite Version.Updated SQLCipher Version.
5.1	Release 5.1.0 (Fermi)	2026-02-11	Updated document version.
5.2	Release 5.2.0 (Interstellar)	2026-03-25	Updated document version.Updated AllowList of IPs for CardEaseXMLAdded Appendix 8 - Axium Device Modes Section
5.3	Release 5.3.0 (Andromeda)	2026-05-01	Updated document version.
5.4	Release 5.3.1 (Andromeda)	2026-06-19	Updated document version.Added support for Ingenico Axum Devices.
5.5	Release 4.1.0(Asteroid)	2026-07-23	Updated document version. Added Surcharging support. Added the Sandbox environment for use with sandbox.nmi.com API keys, and its firewall requirements.Added iOS build flavours
5.6	Release 4.1.1(Asteroid)	2026-07-29	Documented SurchargeNotSupported, returned when surcharging is requested on an unsupported transaction type or for a merchant not enabled for surcharging.
5.7	Release 5.4.0 (Cassiopeia)	2026-07-30	Updated document version. Added Credential on File first store support for SmartPOS integrations. Updated supported feature table to indicate that surcharging is not supported for SmartPOS integrations.
5.8	Release 5.4.1 (Cassiopeia)	2026-08-14	Updated document version.Updated the Cloud Commerce SDK dependencies and AAR names for Tap To Mobile 6.9.0.Documented the getEnvironment() override used to select the Tap To Mobile environment.Deprecated CertificateFingerprint.Clarified surcharge refund behaviour for refunds performed by CardEase reference.

Document Version	Software Version	Date (yyyy-mm-dd)	Summary of Changes
5.9	Release 5.4.2 (Cassiopeia)	2026-10-30	Updated document version. Added receipt printing for Smart POS integrations. Added the Smart POS admin settings to connectAndConfigure and getStatus. Added merchant tipping for VP3350 and Chipper PIN pads, and the Merchant Tipping TMS property. Smart POS currencies now come from the merchant's boarding. Added the CardEntryTimeout parameter for Smart POS. Added card tokens and FundingCardTokens for Smart POS. Added Customer Vault, Credential on File and Incremental Authorization sections.

1. How to Use This Guide

Throughout this guide you will notice a number of visual indicators and styles used to emphasize important information. These are explained below.

Caution



Critical information that must be obeyed to ensure success or avoid significant problems.

Key Point or Important Concept



Key information that you should check you understand fully before continuing.

Helpful Hint



Hints and tips to help you act more effectively or avoid common mistakes.

Reference Material



References to further information outside this guide, such as international standards and useful Internet addresses.

Computer Text

Directories, filenames, commands, variables and the like are presented in-line like this: ChipDnaMobile

…or in code blocks like this:

```
public ChipDnaMobile(Parameters parameters)
```

References and Hyperlinks

Cross-references to other parts of this guide are clickable hyperlinks presented like this:

How to Use This Guide

References to Internet locations (URLs) are clickable hyperlinks presented like this:
www.google.com

2. Document Scope

This document provides a high level description of the API for integrating the Payment Device SDK for iOS/Android into a POS application.

3. What You Will Need

Payment Gateway Account

Register for a gateway account with your partner:

1. Once logged in, click on Settings – Security Keys and follow the on-screen instructions.
2. Your API Key will need to have ‘API’ source permissions only.



Keep the API Key details safe - you need this to configure the Payment Device SDK for processing transactions.

Application Identifier

The integrating application should always utilise the same value. The application identifier cannot be changed once configured on the TMS platform.

The value for this identifier should be a single word or acronym that uniquely identifies the application, for example company name or application name. It should contain only alphanumeric characters in the range A-Z and 0-9 and the same value should be used for the same integrating application across all platforms.

Payment Device SDK for iOS/Android Software

The SDK software is packaged and supplied as .zip archives, one for Android and the other for iOS. The contents are described in Table 1 and Table 2.

Table 1 – Contents of Payment Device SDK for Android .zip archive.

Folder Name	Description
ChipDnaMobileJavaDemo	An example client written in Java, including source code, demonstrating a simple integration using the ChipDNA Mobile SDK.
ChipDnaMobileKotlinDemo	An example client written in Kotlin, including source code, demonstrating a simple integration using the ChipDNA Mobile SDK.
doc	The documentation of the ChipDNA Mobile API.

Folder Name	Description
libs	The ChipDNA Mobile SDK binaries.

Android Flavours

Within the libs folder, there are two further folders, representing the two flavours of the ChipDNA Mobile SDK;

- libs/lite
- libs/full

Lite

The Lite flavour has a minimum OS of Android 10 (SDK version 29) and does not support the use of Tap To Pay.

The ChipDnaMobile jar file has a different name of ChipDnaMobile-lite.jar.

Full

The Full flavour has a minimum OS of Android 12 (SDK version 31) and does support the use of Tap To Pay, requiring the extra dependencies of the Cloud Commerce SDK.



Do not include both SDK flavours in the same project. Choose the one that best fits your integration approach.

Table 2 – Contents of Payment Device for iOS .zip archive.

Folder Name	Description
ChipDnaMobileSwiftDemo	An example client written in Swift, including source code, demonstrating a simple integration using the ChipDNA Mobile SDK.
doc	The documentation of the ChipDNA Mobile API.
ChipDnaMobile/Library	The ChipDNA Mobile SDK binaries and header files.
ChipDnaMobile/ Framework	The ChipDNA Mobile SDK binaries as XCFrameworks

Table 2 shows that two Payment Device SDK variants are now supplied in the iOS release.

- ChipDnaMobile/Library contains the SDK as static libraries and header files.
- ChipDnaMobile/Framework contains the SDK as XCFrameworks.



Do not include both SDK variants in the same project. Choose the one that best fits your integration approach.

IOS Flavours

Within the main folder, there are two further folders, representing the two flavours of the ChipDNA Mobile SDK;

- SDK/lite
- SDK/full

Lite

The Lite flavour has a minimum OS of iOS 17.4 and does not support the use of Tap To Mobile.

Full

The Full flavour has a minimum OS of iOS 18.5 and does support the use of Tap To Mobile, requiring the extra dependencies needed for Tap To Pay on iPhone.

Tap To Mobile Supplement

If you intend to make use of the Payment Device SDK’s Tap To Mobile functionality you should ensure to read the Payment Device SDK for iOS Tap To Mobile Supplement, or Payment Device SDK for Android Tap To Mobile Supplement . While some of the information has been made available in this document the supplement goes into greater detail and was created to help ease the process of adding Tap To Mobile to an integrating application.

4. Transaction Flow

In this section we use a typical example to help you understand how to approach integration of the Payment Device SDK into your payment solution. The majority of payments will be processed in the same way, as shown in the following Transaction Diagram.

Compliance and security policy require that live cards should only be used with devices purchased as production devices. Similarly, test cards should only be used with devices purchased as test devices.

Transaction Diagram

Figure 1 shows the flow of a typical EMV transaction from your perspective as the integrator. For simplicity the diagram assumes that the PIN is correct and no further authentication is required.

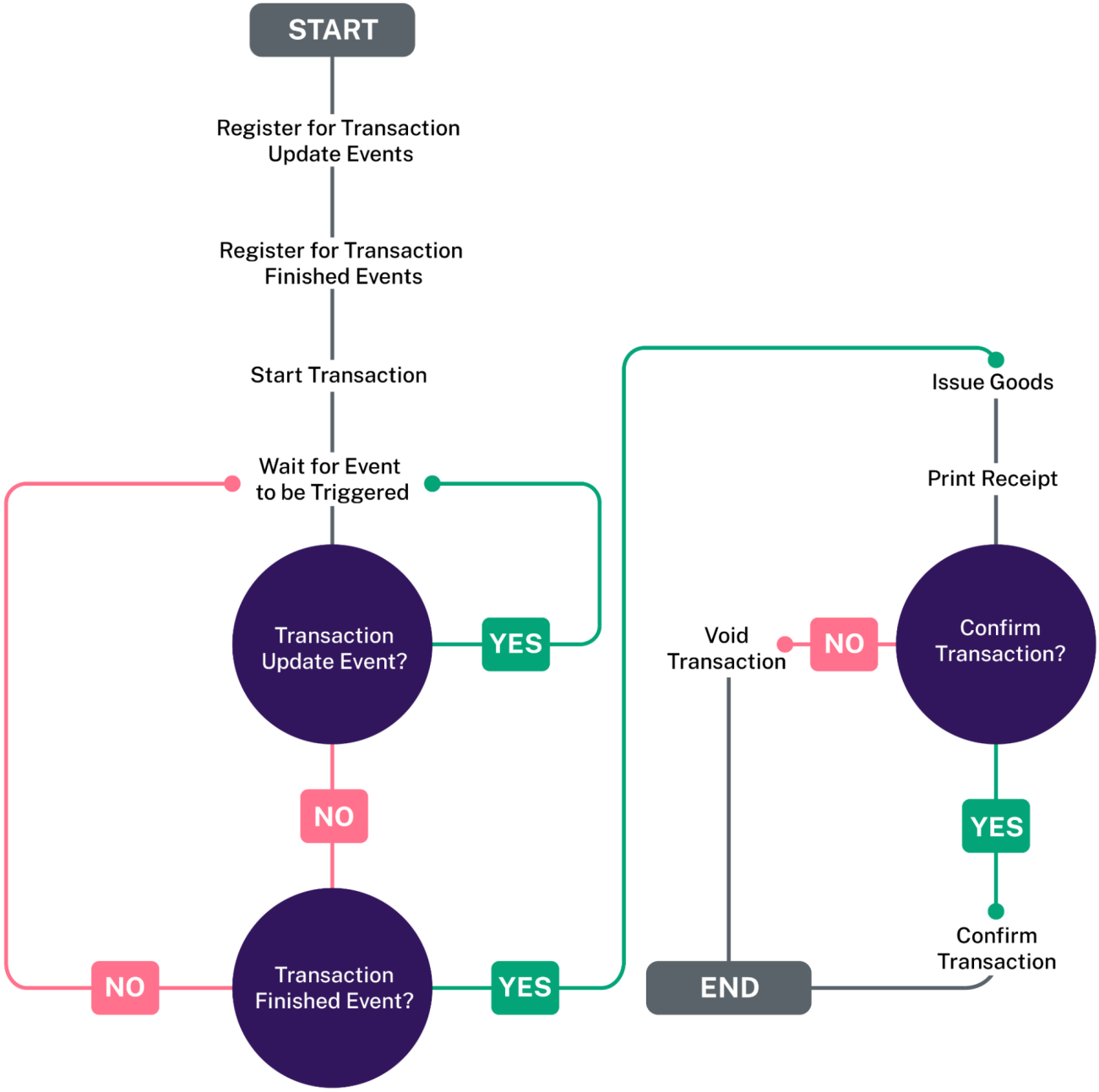


Figure 1 - Flow diagram for a typical EMV transaction.

After calling the startTransaction() method, the Client application may listen for the following events:

Mandatory

- signatureVerification – Fired during a transaction authorization when cardholder signature verification is required.

- transactionFinished – Fired when the transaction is completed on the PIN pad.
- userNotification – BBPOS Chipper 2X BT and ID TECH VP3350 only. Fired when a prompt is required to be shown to the customer.
- applicationSelection – BBPOS Chipper 2X BT and ID TECH VP3350 only. Fired when POS requires the customer to select the card application they wish to complete the transaction with.

Optional or Configuration Dependent

- transactionUpdate – Fired throughout the transaction to update the POS software on the progress of the transaction.
- signatureCapture – Mandatory when digital signature capture is supported. Fired when digital signature needs to be supplied.

To mark an authorized transaction for settlement, call confirmTransaction() at the end of the transaction authorization, after the transactionFinished event is fired.



The Payment Device SDK does not auto-confirm (capture) transactions. Approved transactions must be confirmed for settlement in order to obtain the funds. If the funds for an approved transaction are not settled for reasons such as goods issue failure then the transaction should be voided.

Transaction Result

Once a standalone transaction is complete the result is returned by the SDK. This will be one of the following:

- APPROVED
- DECLINED
- VOIDED

A typical transaction requires:

1. Online authorisation from the acquirer
2. Authorisation by the terminal (including the result of signature verification)

A transaction must be approved in both stages for the transaction result to be approved otherwise it will be declined. The acquirer might approve the transaction partially, in which case the

transaction will be partially approved and the partially approved amount will be returned in the transaction result.

Transaction State

The state of the transaction will be one of the following:

- Uncommitted
- Committed
- Void
- Uncommitted void
- Delayed

If the online authorisation is declined, the transaction will always be declined and its state will be uncommitted.

However, if the online authorisation is approved or partially approved the value of the state depends on whether the transaction is approved by the terminal. If it is approved by the terminal, the result will be approved and the transaction must be committed with a call to confirmTransaction() so the funds for the transaction will be settled or voided using a call to voidTransaction().

If the transaction is approved online but declined by the terminal, the result will be declined and the transaction will be automatically voided.

A transaction state of Delayed is returned when a transaction performs delayed online processing. In this case the transaction never goes online and an encoded request will be returned in the transaction finished event which can be processed using another integration.

5. Configuration

POS Registration

Upon first connection to a device the SDK will generate an identifier for the install of the application. This will be sent to the payment gateway along with details of the selected PIN pad and the API Key and Application Identifier. This information will then be registered on the payment gateway. Should this identifier be lost due to application wipe or re-install this would cause the POS to be re-registered to the payment gateway.



If you haven't already done so, create your API Key. See section Payment Gateway Account for details.



It may be necessary to configure TMS properties specifically for an integrating application. For this reason, an application identifier must be passed to the Payment Device SDK. If you haven't already done so, register an application identifier with the Payment Gateway. See section Application Identifier for details.

Configuration Updates

It is a scheme requirement that the PIN pad can be configured and updated remotely. Any changes in the properties or software used by the PIN pad are updated automatically by the SDK when a connection is opened.

ID TECH VP3350 devices allow for firmware updates to be controlled by the integrator through passing a parameter into **ConnectAndConfigure** or **RequestTmsUpdate**. See section Configuration and Utility Methods for details on usage. In the case of all PIN pads, if a firmware update has been started it will automatically finish the update on **ConnectAndConfigure.**

The SDK determines whether updates are required and carries out the required updates when connecting to a PIN pad. However it is advised that functionality to perform a manual update is provided in the integrating application.

Two types of updates are carried out:

- Regular (status) updates. It is a scheme requirement that a terminal can be remotely disabled and these updates ensure the SDK checks the status of the terminal regularly, most commonly required every 24 hours.
- Terminal (configuration) updates. These updates download the properties for the SDK configured on the TMS servers. It is a scheme requirement that the terminal performs an update at regular intervals, most commonly required every 45 days.

An update can be required for a number of reasons, for example:

- The maximum number of hours since the last update has passed.
- The maximum number of days since the last terminal update has passed.
- The API Key has changed.
- The selected PIN pad has changed
- The version of the SDK has changed
- An update was requested by the gateway during a transaction.
- The Application Identifier (App ID) has changed

Environments

The SDK can be pointed at different platforms using the environment property set

via setProperties(). The available environments are:

- Live
- Production platform. Use a production API key for live transactions. |
- Staging
- Test platform. Use a test API key for integration and testing.
- Sandbox
- Use this environment when integrating with a sandbox.nmi.com API key. Identical to Staging, except that gateway calls are routed to sandbox.nmi.com.

For Tap To Mobile the Cloud Commerce environment is selected separately, by overriding getEnvironment() in your ChipDnaApplication subclass. Return ParameterValues.LiveEnvironment for production or ParameterValues.TestEnvironment for testing; it defaults to ParameterValues.LiveEnvironment. This value should agree with the

environment supplied to setProperties(). See the Payment Device SDK for Android Tap To Mobile Supplement.

6. Modes Of Operation

Payment Device SDK offers flexible transaction processing options. It can be configured to handle Tap to Mobile transactions independently, process transactions through an external payment device, or operate in a combined mode. In the combined mode, transactions can be initiated simultaneously on both Tap to Mobile and an external payment device, allowing for versatile and seamless payment processing.

Configuration

During calls to connectAndConfigure you can request which mode the Payment Device SDK will operate in using the following parameters.

Tap To Mobile-Only Environment

- Include ParameterKeys.TapToMobilePOI with a value of ParameterValues.TRUE.
- Include ParameterKeys.PaymentDevicePOI with a value of ParameterValues.FALSE.

This configuration stops the Payment Device SDK from attempting to set up an external payment device.

Tap To Mobile and Payment Device Environment

- Include ParameterKeys.TapToMobilePOI with a value of ParameterValues.TRUE.
- Include ParameterKeys.PaymentDevicePOI with a value of ParameterValues.TRUE. This configuration enables the use of both Tap To Mobile and external payment devices.

Default Payment Device Behavior

If no POI Parameter key is provided, the SDK will default to enabling the external payment device and Tap To Mobile transactions will not be offered.

Transaction processing

Once configured you can make use of the ParameterKeys.TransactionPOI parameter key to select which POI the transaction is started on.

Accepted Parameter Values

- ParameterValues.PaymentDevice: Specifies the use of a physical payment device for the transaction.

ParameterValues.TapToMobile: Specifies the use of the Tap To Mobile functionality for the transaction.

Note: when a single POI is configured, that POI will be used by default. When both have been configured the PaymentDevice will be preferred.



Android requires that the integrating app must register to the RequestActivityListener for transaction processing.

7. Configuration and Utility Methods

Parameters

The request and response data in the Payment Device SDK API methods is set in a Parameters object. The Parameters object is a collection of name-value pairs representing the data sent to or received from the SDK.

Initialization

```
Parameters initialize(Context, Parameters)
```

Your application interacts with the Payment Device SDK using a method in the ChipDnaMobile object instance. Before beginning the messaging process, you must initialize a new ChipDnaMobile instance.

This method returns a Parameters object with the result of the initialization. Any errors encountered will also be reported in this object.

SetProperties

```
Parameters setProperties(Parameters)
```

Call setProperties() to set configuration settings for the Payment Device SDK. This method is synchronous and results are returned in the parameters.

setProperties() Is also used in the SDK to set:

- API Key
- Application Identifier
- PIN pad name

ConnectAndConfigure

```
Parameters connectAndConfigure(Parameters)
```

After initialization and the correct properties are set, a call to this method will connect to the PIN pad and if needed downloads configuration data from TMS and configures the device in preparation for performing transactions.

When the connectAndConfigure() method call is successful, asynchronous updates are provided in the configurationUpdate and connectAndConfigureFinished callbacks.

The connectAndConfigure() can also be used to perform a TMS update by including the ForceTmsUpdate parameter with a value of TRUE.

Firmware Updates

The connectAndConfigure() can also be used to perform a Firmware update. To control when an update occurs include the ApplyFirmwareUpdate parameter. If no parameter is supplied it will default to applying a firmware update straight away if new firmware is available. With a value of FALSE it can be delayed until an update is critical, a time frame defined in FirmwareUpdateConfiguration TMS property. When an update is critical, connectAndConfigure() will fail with a FirmwareUpdateRequired error code. This firmware update control feature is currently only supported on ID TECH VP3350 PIN pads, where an update is automatically applied if available for all other PIN pads.

Tap To Mobile

The connectAndConfigure method enables the setup and configuration of both Tap To Mobile and external payment devices. The behavior of this method is determined by the parameters provided during the call. Passing in the TapToMobilePOI with the value of TRUE will prompt ChipDNA Mobile to configure its Tap To Mobile capabilities and allow the processing of Tap To Mobile transactions.

If you wish to set up ChipDNA Mobile so only Tap To Mobile is supported you must pass in the TapToMobilePOI with the value of TRUE and PaymentDevicePOI with the value of FALSE.

For more information on this see the Payment Device SDK for iOS Tap To Mobile Supplement documentation supplied with the release under Modes Of Operation.

Smart POS Admin Settings

On Smart POS devices, connectAndConfigure() can also change the settings on the terminal's Admin Screen. Include any of the following parameters:

- ParameterKeys.KioskModeEnabled: TRUE or FALSE. Kiosk mode locks the terminal to the Transaction Initiator UI. Kiosk mode keeps the Transaction Initiator UI in front whenever no transaction is running, so another app on the same device cannot stay in the foreground. Do not turn it on if your point-of-sale app runs on the Axium device itself.
- ParameterKeys.NotificationPanelAllowed: TRUE or FALSE. Whether the Android notification panel can be pulled down. A value set through connectAndConfigure applies whether or not kiosk mode is on. After the setting is changed on the Admin Screen, it applies only while kiosk mode is on, until it is next set through connectAndConfigure.
- ParameterKeys.NewAdminPasscode: a new administrator passcode for the Admin Screen, of 6 to 12 digits. The current passcode is not needed. Pass it only when changing the passcode.

The settings need Transaction Initiator UI v1.4.0 or later. Only pass them to Smart POS: on a standard payment device, or with a KioskModeEnabled or NotificationPanelAllowed value other than TRUE or FALSE, or an empty NewAdminPasscode, connectAndConfigure returns Result FALSE straight away without connecting. Otherwise wait for connectAndConfigureFinished before starting a transaction. See the Axium Smart POS Supplement for how the settings are applied and which errors leave the terminal connected.

GetAvailablePinPads

```
Parameters getAvailablePinPads(Parameters)
```

Call getAvailablePinPads() for the Payment Device SDK to search for available PIN pads.

getAvailablePinPads() is asynchronous. The results of the PIN pad search will be provided in the onAvailablePinPads callback.

The request parameters can be used to indicate which communication protocol to use (e.g., in Android it is possible to search for PIN pads on Bluetooth Classic, Bluetooth Low Energy (BLE), USB or any combination of communication protocols.)

The length of time that the SDK will scan for BLE devices is customizable. Increasing the Bluetooth Low Energy scanning time will result in a higher likelihood of the BLE device being discovered.

GetStatus

```
Parameters getStatus(Parameters)
```

Call getStatus() to get the current status of different components of the Payment Device SDK in a single call. If no parameters are specified, the status of all the different components is checked. The request parameters can be used to specify which components need to be checked.

getStatus() is a synchronous method which will block and return the requested status data in the response parameters. Information returned includes offline queue status, current properties, the SDK version, firmware update status, payment device information, and payment platform availability.

Smart POS Admin Settings Status

When a Smart POS device is connected, getStatus() also returns ParameterKeys.KioskModeEnabled, ParameterKeys.NotificationPanelAllowed, ParameterKeys.KioskSettingsApplied and ParameterKeys.AdminPasscodeIsDefault. These are not returned when the terminal is not connected, or its Transaction Initiator UI is earlier than v1.4.0.

Identifiers

The Payment Device SDK will return the following identifiers during calls get status when available:

- ParameterKeys.POSGUID
- An identifier representing the Point of Sale Device. This value is used to identify the Point of Sale with the payment gateway.
- ParameterKeys.TapToMobilePOIIdentifier
- An identifier representing the Tap To Mobile Point of Interaction. This value is used to identify this instance of Tap To Mobile to the payment gateway.



It’s highly recommended to make these identifiers accessible to application users to facilitate troubleshooting for specific devices.

GetTransactionInformation

```
Parameters getTransactionInformation(Parameters)
```

Call getTransactionInformation() to get the current information corresponding to the specified transaction. This information includes, among other items, the transaction result, transaction time, whether it has been confirmed or not.

GetMerchantData

```
Parameters getMerchantData(Parameters)
```

Call getMerchantData() to get the current information corresponding to the configured merchant accounts. This information includes, the currencies supported, transaction types supported and the merchant’s name and number.

RequestTmsUpdate

```
Parameters requestTmsUpdate(Parameters)
```

Use requestTmsUpdate() to force an immediate TMS update.

requestTmsUpdate() is not required as part of the normal workflow as ConnectAndConfigure will automatically perform a TMS update. This method requires a connection to the PIN pad to be established before it can be called.

The requestTmsUpdate() can also be used to perform a Firmware update. To control when an update occurs include the ApplyFirmwareUpdate parameter. If no parameter is supplied it will default to applying a firmware update straight away if new firmware is available. With a value of FALSE it can be delayed until an update is critical, a time frame defined in FirmwareUpdateConfiguration TMS property. When an update is critical, it will prevent actions requiring the PIN pad to be performed, these are startTransaction() and getCardDetails(). This firmware update control feature is currently only supported on ID TECH VP3350 PIN pads, where an update is automatically applied if available for all other PIN pads.

If requestTmsUpdate() method call is successful the result of the TMS update result is provided in the tmsUpdate event.

Disconnect

```
Parameters disconnect(Parameters)
```

Call this method to disconnect from any connected PIN pads.

Dispose

```
Parameters dispose(Parameters)
```

Call this method to dispose of the ChipDnaMobile instance. This will release resources including closing database connections and disconnecting from any connected PIN pads. After dispose() is called, initialize() must be called to use the Payment Device SDK again.

8. Configuration and Utility Events

AvailablePinPads Callback

When getAvailablePinPads() has been successfully called, the Payment Device SDK fires a onAvailablePinPads callback when the update is finished with the result of the request. The result will contain a collection of PIN pads that are paired with the mobile device.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addAvailablePinPadsListener(
    IAvailablePinPadsListener listener
)
void removeAvailablePinPadsListener(
    IAvailablePinPadsListener listener
)
```

iOS

```
void addAvailablePinPadsTarget:(id)self
     action:(SEL)@selector(onAvailablePinPads:);
void removeAvailablePinPadsTarget:(id)self;
```

ConnectAndConfigureFinished Callback

When connectAndConfigure() has been successfully called, the Payment Device SDK fires a connectAndConfigureFinished event with the result of the request. This event will always be fired and the parameters will contain data to indicate if the SDK is ready to process transactions. If unsuccessful, the parameters will contain the errors encountered.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addConnectAndConfigureFinishedListener(
    IConnectAndConfigureFinishedListener listener
)

void removeConnectAndConfigureFinishedListener(
    IConnectAndConfigureFinishedListener listener
)
```

iOS

```
void addConnectAndConfigureFinishedTarget:(id)self
     action:(SEL)@selector(onConnectAndConfigureFinished:)

void removeConnectAndConfigureFinishedTarget:(id)self
```

ConfigurationUpdate Callback

When connectAndConfigure() has been successfully called, and before connectAndConfigureFinished event is fired. The Payment Device SDK fires configurationUpdate events as it progresses through the configuration process.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addConfigurationUpdateListener(
    IConfigurationUpdateListener listener
)

void removeConfigurationUpdateListener(
    IConfigurationUpdateListener listener
)
```

iOS

```
void addConfigurationUpdateTarget:(id)self
     action:(SEL)@selector(onConfigurationUpdate:)

void removeConfigurationUpdateTarget:(id)self
```

DeviceUpdate Callback

The Payment Device SDK fires deviceUpdate events to report notification about the PIN pad. Events include connection, disconnection, initialization, reboot events.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addDeviceUpdateListener(
    IDeviceUpdateListener listener
)

void removeDeviceUpdateListener(
    IDeviceUpdateListener listener
)
```

iOS

```
void addDeviceUpdateTarget:(id)self
     action:(SEL)@selector(onDeviceUpdate:)

void removeDeviceUpdateTarget:(id)self
```

TmsUpdate Callback

When requestTmsUpdate() has been successfully called, the Payment Device SDK fires a tmsUpdate callback result of the TMS update request.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addTmsUpdateListener(
    ITmsUpdateListener listener
)

void removeTmsUpdateListener(
    ITmsUpdateListener listener
)
```

iOS

```
void addTmsUpdateTarget:(id)self
     action:(SEL)@selector(onTmsUpdate:)

void removeTmsUpdateTarget:(id)self
```

FirmwareUpdate Callback

Delayed Firmware Updates

When connectAndConfigure() or requestTmsUpdate() has been successfully called, and before connectAndConfigureFinished event is fired. The Payment Device SDK fires a firmwareUpdate callback to indicate that a firmware update is available for the currently configured device. This will contain a FirmwareUpdateStatus object indicating when the firmware update is required to take place by. To initiate a firmware update, call connectAndConfigure() or requestTmsUpdate() with the ParameterKeys.ApplyFirmwareUpdate set to TRUE or not present.

This is currently only supported on ID TECH VP3350 PIN pads.

Firmware Update Progress

The firmwareUpdate listener will be returned throughout the firmware update process if a firmware update takes place; after connect and configure and request tms update if there is a firmware update available, and during a firmware update to indicate its progress.

This is currently supported on Miura and ID TECH VP3350 PIN pads.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addFirmwareUpdateListener(
    IFirmwareUpdateListener listener
)

void removeFirmwareUpdateListener(
    IFirmwareUpdateListener listener
)
```

iOS

```
void addFirmwareUpdateTarget:(id)self
     action:(SEL)@selector(onFirmwareUpdate:)

void removeFirmwareUpdateTarget:(id)self
```

9. Payment Methods

StartTransaction

```
Parameters startTransaction(Parameters)
```

Call startTransaction() to initiate a transaction. This can be used to start a Sale, Refund or Account Verification transaction. The request accepts various parameters including amount, transaction type, unique reference, whether to enable/disable gratuity and currency.

When the startTransaction() method is successful, asynchronous updates are provided in the following callbacks; transactionUpdate, userNotification, applicationSelection, signatureVerification, and transactionFinished.

Transaction POI - Tap To Mobile

Using the ParameterKeys.TransactionPOI parameter allows you to dictate whether the transaction is run on an external device or as a Tap To Mobile transaction.

When a single POI is Configured (either PaymentDevice or TapToMobile), it will be automatically selected when starting a transaction.

When both payment types are configured the payment device will be used by default unless the TransactionPOI parameter explicitly specifies Tap To Mobile.

Please note that Tap To Mobile transactions are currently online only as an auto confirmed sale. Therefore the following is not supported when running a Tap To Mobile Transaction:

- TransactionType
- Refund
- AccountVerification
- DelayOnlineProcessing
- TRUE
- PANKeyEntry
- TRUE
- TippingType
- OnDeviceTipping
- EndOfDayTipping
- BothTipping
- AutoConfirm
- FALSE



Please also note that before initiating a transaction using Tap To Mobile, make sure the ***Request Activity Callback*** has been implemented.

Tipping

Merchant tipping is performed by passing the tip in the TipAmount parameter of startTransaction(), in the currency's minor units. The tip is added to the transaction amount. TipAmount is only allowed on a Sale; leave it out when there is no tip.

TipAmount is supported on Tap To Mobile, Smart POS and, on Android, the ID TECH VP3350 and BBPOS Chipper PIN pads. Other PIN pads, such as Miura and Datecs, and all PIN pads on iOS, return TipAmountNotAllowed.

On the VP3350 and Chipper, merchant tipping must be enabled in TMS (see the TMS Properties appendix), and the amount plus the tip must not exceed the maximum transaction amount configured in TMS. TipAmount is rejected with:

- MerchantTippingNotSupported: merchant tipping is not enabled in TMS.
- TipAmountInvalid: the tip is not a whole number of minor units.
- TipAmountNotAllowed: the transaction is not a Sale, uses delayed online processing, or also includes TippingType or OnDeviceTippingPrompt.
- AmountTooLarge: the amount plus the tip exceeds the maximum transaction amount configured in TMS.

Any surcharge is calculated on the amount before the tip.

Tap To Mobile also needs merchant tipping enabled in TMS; see the Tap To Mobile Supplement for its errors. For Smart POS, see the Axium Smart POS Supplement.

Card Entry Timeout

On Smart POS devices, the CardEntryTimeout parameter sets how many seconds to wait for a card to be presented before the card entry prompt times out, as a whole number from 1 to 3600. It is accepted by startTransaction() and getCardDetails(). Any other value returns CardEntryTimeoutInvalid. If it is not supplied, or is blank, the device's configured default is used. It needs Transaction Initiator UI v1.4.0 or later; earlier versions ignore it. See the Axium Smart POS Supplement.

Auto Confirm

The AutoConfirm parameter with the value TRUE should be passed into startTransaction() in order to request that an auto confirm transaction is started.

During an auto confirm transaction the SDK makes an attempt to complete the second stage of a transaction before the transactionFinished callback has been fired. This means that a subsequent call to confirmTransaction() or voidTransaction() is not required. Attempting to confirm a transaction which has already been completed will result in an error.

If the SDK fails to complete the second stage of the transaction for whatever reason the TransactionState returned in the transactionFinished event will be either Uncommitted or UncommittedVoid along with an appropriate error. The integrating application must then take

responsibility for ensuring the second stage of a transaction is performed by calling either confirmTransaction() or voidTransaction().

PAN Key Entry

The PANKeyEntry requests a PAN key entry transaction is started. The card details are keyed into the PIN pad and are used for Card Not Present transactions. If the terminal is configured to require AVS during PANKeyEntry the numerics found in the first line of the customers address and their zip code will also need to be keyed into the PIN pad. The PANKeyEntry value should be set to TRUE or FALSE.

It's important to ensure that the acquirer is expecting to accept Card Not Present transactions on the associated merchant account.

Account Verification

An Account Verification transaction is a transaction that does not result in any transfer of funds, but is used to validate that the card is associated with an active account.

The TransactionType value should be set to AccountVerification to enable the Account Verification transaction. The transaction only requires a currency and user reference. The amount should not be included as part of the parameters.

Delay Online Processing

In Android it is possible to run a transaction with delayed online processing and process the request later via a different integration. To run a transaction with delayed online processing the DelayOnlineProcessing parameter key should be supplied with the value set to TRUE. No amount should be supplied when calling startTransaction. A delayed online processing transaction will complete to the point of going online at which point the transaction will finish and an encoded request will be returned in the transaction finished event.

Surcharging

Surcharging applies a merchant-permitted fee to a card-present transaction. To opt in, pass the Surcharge parameter with the value TRUE into startTransaction()(it defaults to FALSE). The SDK then fetches the merchant's surcharge limits from the gateway and fires the surcharge callback, and the amount to apply is returned to the SDK with a call to continueSurcharge().

Surcharging is US-only, online-only, and available for Sale, Auth and Linked Refund transactions. It must be enabled for the merchant account on the gateway, and a surcharge callback observer must be registered before startTransaction() is called, otherwise SurchargeObserverRequired is returned. Passing the Surcharge parameter into startTransaction() for a transaction type that does not support surcharging returns SurchargeNotSupported. SurchargeNotSupported is also returned when the merchant account is not enabled for surcharging on the gateway, for example a non-US merchant, or when the SDK is operating offline.

Surcharge limits are based on the location where the transaction takes place. This location can be supplied using the TransactionAddressLine1, TransactionAddressLine2, TransactionAddressCity, TransactionAddressState, TransactionAddressZip and TransactionAddressCountry parameters; if none are supplied the merchant address on file is used. If any address component is supplied the full address must be provided, and TransactionAddressCountry is always required, if not supplied then the transaction will fail with the error: TransactionAddressMissing



The integrating application is responsible for the accuracy of the transaction address. It should be the location where the transaction takes place, for example the store or job-site address for a card-present sale, or the customer's billing address for e-commerce and recurring payments.. An inaccurate address may result in an incorrect surcharge limit or eligibility response

When a surcharge has been applied, the SurchargeAmount and the SubtotalAmount (the amount before surcharge) are provided, in minor units, in both the transactionFinished callback and the response from getTransactionInformation. In both cases the Amount reflects the total inclusive of the surcharge. SurchargeAmount is omitted when no surcharge was applied and its absence should be treated as no surcharge.

To refund a surcharged transaction, the portion of the surcharge to be refunded should be supplied in the SurchargeAmount parameter to linkedRefundTransaction(), in minor units and pro-rated in the case of a partial refund. Where the original transaction is held locally, for example a refund performed by UserReference, supplying a surcharge amount for an original transaction that carried none returns SurchargeRefundNotSupported. Where the refund is performed by CardEase reference the original transaction is not available locally, it may have been taken on another device, so the supplied surcharge amount is sent to the gateway to be validated there.

Surcharging does not support partial approvals. If the acquirer partially approves a surcharged transaction, the SDK voids the transaction rather than completing it for the partially approved amount.

The surcharge percentage is stored with the transaction data so that surcharge limits can be evaluated correctly when an approved transaction is later confirmed for a lower amount. When a lower amount is confirmed with confirmTransaction(), the stored percentage is applied to the total and surcharge amount being confirmed.

Surcharging is not supported for SmartPOS integrations.

One way Card Tokens

In Android it is possible to request Card Tokens for a transaction. Card Tokens are consistent, non-payment-specific tokenization of card PANs. These tokens facilitate use cases such as fare aggregation, transaction matching, and other application-specific scenarios.

To Request Card Tokens:

- Include the FeatureTokens parameter key in your startTransaction() transaction request.
- Generate an XML String of the Feature Tokens using the following serializer method provided by the Payment Device SDK:
- ChipDnaMobileSerializer.serializeFeatureTokens(List)
- Pass this XML string as the value for the FeatureTokens parameter in your startTransaction() request.

Upon successful transaction processing, the generated CardTokens will be returned in the TransactionFinished callback as an XML String, this can be deserialized using ChipDnaMobileSerializer.deserializeCardTokens(String) .

If the FeatureTokens passed in are invalid, FeatureTokensInvalid is returned in the startTransaction() response.

Card Tokens on Smart POS

Card tokens are also supported on Smart POS, with Transaction Initiator UI v1.4.0 or later. CardTokens is returned in the TransactionFinished callback and by getTransactionInformation(). When the card is a mobile wallet or other device card number, and the authorization supplies the funding card, tokens for the funding card are also returned in FundingCardTokens, in the same format as CardTokens. For an offline or deferred Smart POS transaction, the tokens are generated when the transaction is processed online; retrieve them with getTransactionInformation().

Obtaining Feature Tokens:

Feature Tokens are generated and digitally signed by the Payment Device Gateway. Integrators should please contact the support team or account manager to obtain the required Feature Tokens, including details about the necessary methods, keys, or configuration information.

Customer Vault

The Customer Vault stores a customer's card on the payment gateway so that it can be charged later without the card being present. To store the card used in a transaction, include these parameters in startTransaction():

- CustomerVaultCommand: add-customer to create a new vault record, or update-customer to replace the card on an existing record.
- CustomerVaultId: the vault record to use, for example the record to update with update-customer.

Each value can be up to 199 characters. A longer value returns CustomerVaultCommandInvalid or CustomerVaultIdInvalid.

The vault record's identifier is returned as CustomerVaultId in the TransactionFinished callback. Customer Vault is supported on iOS and Android PIN pads, on Smart POS and, on Android, on Tap To Mobile. Later charges against a vault record are made through the payment gateway, not the SDK.

Credential on File

Credential on File marks a Sale as the first time a card is stored for future charges, such as recurring billing, installments or unscheduled charges, and records the reason. Include these parameters in startTransaction():

- CredentialOnFileFirstStore: ParameterValues.TRUE. The Credential on File parameters are only sent for a Sale with this set to TRUE; otherwise they are ignored.
- CredentialOnFileReason (optional): the reason the card is stored. Values can be ParameterValues.ReasonUnscheduled, ReasonInstallment, ReasonIncremental, ReasonResubmission, ReasonDelayedCharge, ReasonReAuth or ReasonNoShow. On Smart POS, ReasonRecurring is also accepted. Any other value returns CredentialOnFileReasonInvalid.

The first store is recorded as initiated by the cardholder. Credential on File is supported on iOS and Android PIN pads, on Smart POS and, on Android, on Tap To Mobile. Later charges using the stored credential are made through the payment gateway, not the SDK.

Incremental Authorization

An incremental authorization lets the authorized amount grow before the transaction is captured, for example for hotel stays, car rental or EV charging. It is available on iOS and Android PIN pads and on Smart POS. The merchant's processor must support incremental authorization.

The SDK performs the initial authorization. Call startTransaction() for a Sale with:

- CredentialOnFileFirstStore set to ParameterValues.TRUE.
- CredentialOnFileReason set to ParameterValues.ReasonIncremental.

Do not use AutoConfirm. The TransactionFinished callback reports the approved initial authorization with a TransactionState of Uncommitted. Unlike other uncommitted transactions, do not confirm it with confirmTransaction(); store its CardEaseReference and TransactionId, and capture it through the payment gateway.

Increasing the authorization, capturing it and voiding it are done through the payment gateway, not the SDK. Every increase and the capture refer to the initial authorization's references. For the payment gateway's incremental authorization guide, contact the support team. Incremental authorization is not available on Tap To Mobile, which only performs auto confirmed sales.

For example, an initial authorization of 10.00 followed by increases of 15.00 and 20.00 gives an authorized total of 45.00, captured in one capture.

- Each increase is for the additional amount only, and refers to the initial authorization, not the previous increase.
- The capture can be for any amount up to the total of the approved authorizations. Capturing less is allowed.
- If an increase is declined, the rest of the chain stays valid, and only the approved amounts can be captured.
- To cancel before capture, void the authorization. Once the initial authorization is voided, no more increases can be added.
- The authorization must be captured before it can settle.

ContinueSignatureVerification

```
Parameters continueSignatureVerification(Parameters)
```

Call continueSignatureVerification() with the result of the signature verification. If a digital signature has been captured it should be sent to the Payment Device SDK in this request.

continueSignatureVerification() method may only be called during a transaction after a onSignatureVerification callback has been triggered.

ContinueSignatureCapture

```
Parameters continueSignatureCapture(Parameters)
```

Call continueSignatureCapture() when the merchant has completed the signature capture process and the digital signature data can be provided.

continueSignatureCapture() method may only be called during a transaction after a onSignatureCapture() callback has been triggered.

ContinueSurcharge

```
Parameters continueSurcharge(Parameters)
```

Call continueSurcharge() from the surcharge callback to tell the SDK whether to apply a surcharge, once the callback has indicated ResponseRequired is TRUE. Set the Result parameter to TRUE to apply a surcharge, or FALSE to apply none; Result is required whenever a surcharge can be applied.

When applying a surcharge, supply the amount in the SurchargeAmount parameter in minor units. The amount must be zero or greater and must not exceed the maximum returned in the callback. If the callback indicated OperatorPinRequired was TRUE, the OperatorPin parameter must also be supplied; an incorrect PIN may be retried, and once retries are exhausted the surcharge is declined and the transaction terminates.

This method is synchronous and returns a Parameters object containing the Result and, if present, any Errors, for example SurchargeAmountExceedsMaximum, in which case continueSurcharge() may be called again with a valid amount.

ConfirmTransaction

```
Parameters confirmTransaction(Parameters)
```

To finalize an approved transaction which is not using the auto confirm feature or has failed to be auto confirmed you must call confirmTransaction() so that the transaction will be settled.

You do not have to call confirmTransaction() immediately after the transaction has been approved. For example, you may want to authorize multiple cards for a single purchase, in which case you would call confirmTransaction() for each approval after all transactions have been authorized.

ConfirmTransaction() requires the unique reference that was provided in StartTransaction() to identify and confirm an authorized transaction.

This method is synchronous, when successful the response will contain the result Approved or Declined and receipt data otherwise the response will contain the errors reported while attempting to confirm the transaction.

Completion requests are processed online and in real-time. If the transaction result is declined it is the responsibility of the integrating application to retry until it is approved.

VoidTransaction

```
Parameters voidTransaction(Parameters)
```

To void a transaction before settlement, but after it has been approved or confirmed so that funding does not take place, call voidTransaction().

voidTransaction() requires the unique reference that was provided in startTransaction() to identify and void a previously authorized transaction.

This method is synchronous, when successful the response will contain the result Approved or Declined and receipt data otherwise the response will contain the errors reported while attempting to void the transaction.

The Payment Device SDK may be configured to provide offline support so it continues to process confirms and voids when connection to the payment gateway has been lost. This provides storage capability of transaction data and processes completion requests once communications are restored.

By default the SDK is configured to be online only so void requests are processed online and in real-time. If the transaction result is declined it is the responsibility of the integrating application to retry until it is approved.

LinkedRefundTransaction

```
Parameters linkedRefundTransaction(Parameters)
```

The Client can call linkedRefundTransaction() to refund all or part of a previously approved and confirmed transaction. The method returns a Parameters object instance containing a parameter list that includes the transaction result, receipt data and errors if they exist. When performing a Cash or Cheque Linked Refund, PaymentMethod ParameterKey should be supplied with the value of ParameterValue of Cash or Cheque respectively, if the PaymentMethod has not been supplied the PaymentMethod will default to ParameterValue Card.

This method is synchronous, when successful the response will contain the result Approved or Declined and the transaction information otherwise the response will contain the errors reported while attempting to refund the transaction.



When performing a Linked Refund with Tap To Pay, CardEaseReference ParameterKey is not supported as a reference to the transaction to be refunded. Only UserReference can be used.

TerminateTransaction

```
Parameters terminateTransaction(Parameters)
```

To cancel a transaction in progress call terminateTransaction(). If the transaction has finished before terminateTransaction() is called and the result is Approved then voidTransaction() can be used to cancel the transaction.

ProcessReceipt

```
Parameters processReceipt(Parameters)
```

Call processReceipt() to send a receipt. The request parameters are used to specify properties for processing receipt (e.g., email, SMS or printer) and also specifying the destination.

On Smart POS devices, processReceipt() prints the receipt on the device's integrated printer when PrintReceipt is TRUE. Printing needs ARC 23.06.01 or later and Transaction Initiator UI v1.4.0 or later. Email and SMS receipts are not supported on Smart POS and return ReceiptSourceEmailNotSupported or ReceiptSourceSMSNotSupported. A Result of TRUE means the print request was accepted. The outcome is delivered to the ProcessReceiptFinished callback as ReceiptPrintResult. See the Axium Smart POS Supplement for requirements and errors.

10. Payment Events

TransactionUpdate Callback

The Payment Device SDK fires transactionUpdate events as the customer progresses through the transaction. Each transaction update event describes the action that triggered the event, including EMV commands and data communication. The POS application can react to each event as needed, such as to update a display.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addTransactionUpdateListener(
    ITransactionUpdateListener listener
)

void removeTransactionUpdateListener(
    ITransactionUpdateListener listener
)
```

iOS

```
void addTransactionUpdateTarget:(id)self
     action:(SEL)@selector(onTransactionUpdate:)

void removeTransactionUpdateTarget:(id)self
```

CardDetails Callback

When startTransaction() has been successfully called, the Payment Device SDK fires cardDetails events when the details are available. When getCardDetails() has been successfully called, the SDK fires a cardDetails event when the process is finished (instead of a transactionFinished event).

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addCardDetailsListener(
    ICardDetailsListener listener
)
void removeCardDetailsListener(
    ICardDetailsListener listener
)
```

iOS

```
void addCardDetailsTarget:(id)self
     action:(SEL)@selector(onCardDetails:)

void removeCardDetailsTarget:(id)self
```

Request Activity Callback

To perform a transaction using Tap To Pay, an activity is required to be passed into the Payment Device SDK. This is so that the SDK can access NFC permissions to prompt and perform a card read. It is not necessary to create a new Activity to parse in, the integrating application can use the current activity.

As the activity is being used to prompt for card Tap, the activity must adhere to the EMVCo Contactless Symbol and supported Card Brand images requirements. The requirements around how these are to be used and displayed are in Appendix 1 - Contactless Symbol Reproduction Requirements.

To enable contactless payments TapToMobile, the integrating application must:

- Register a RequestActivityListener before calling startTransaction. This is necessary because the SDK needs an Activity to handle the NFC prompt for card reads.
- When the SDK requires an Activity, it triggers RequestActivityListener.onRequestActivity. The application must respond by calling continueRequestedActivity with an appropriate Activity.
- Note: The provided Activity must remain in the foreground and active until a CardTapped transaction update is received. If the Activity is destroyed or moved to the background too soon, the transaction may fail.
- Once the transaction proceeds, the SDK emits transaction events, including CardEntryPrompted, which enables card reading.
- When the transaction completes, the application is notified via onTransactionFinishedListener.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addRequestActivityListener(
    IRequestActivityListener listener
)

void removeRequestActivityListener(
    IRequestActivityListener listener
)

void clearAllRequestActivityListener()

Parameters continueRequestedActivity(
    Activity activity
)
```

TransactionFinished Callback

After startTransaction() has been successfully called, the Payment Device SDK fires a transactionFinished callback when the transaction is finished. If the transaction is finished after the cardDetails callback is triggered, the transcationFinished parameters contains a parameter which will include the transaction result Approved or Declined and a receipt can be issued for the completed transaction. If the transaction is finished before the cardDetails callback, the parameters contain only the Errors parameter indicating the transaction has been terminated.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addTransactionFinishedListener(
    ITransactionFinishedListener listener
)

void removeTransactionFinishedListener(
    ITransactionFinishedListener listener
)
```

iOS

```
void addTransactionFinishedTarget:(id)self
     action:(SEL)@selector(onTransactionFinished:)

void removeTransactionFinishedTarget:(id)self
```

Transactions may be returned with the state **Uncommitted**, in which case it is the responsibility of the integrating application to call **confirmTransaction()**. Approved transactions must be confirmed for settlement in order to obtain the funds. If the funds for an approved transaction are not settled for reasons such as goods issue failure then the transaction should be voided.

ProcessReceiptFinished Callback

When processReceipt() has been successfully called, the Payment Device SDK fires a onProcessedReceiptFinished callback with the result of the request.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addProcessReceiptFinishedListener(
    IProcessReceiptFinishedListener listener
)

void removeProcessReceiptFinishedListener(
    IProcessReceiptFinishedListener listener
)
```

iOS

```
void addProcessReceiptFinishedTarget:(id)self
     action:(SEL)@selector(onProcessReceiptFinished:)

void removeProcessReceiptFinishedTarget:(id)self
```

SignatureVerification Callback

The Payment Device SDK fires a signatureVerification event, during a transaction, when the merchant needs to verify the cardholder signature. Call continueSignatureVerification() to set the signature verification result and continue the transaction. If digital signature is captured, the data should be sent to the SDK in a parameter in the continueSignatureVerification() method.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addSignatureVerificationListener(
    ISignatureVerificationListener listener
)

void removeSignatureVerificationListener(
    ISignatureVerificationListener listener
)
```

iOS

```
void addSignatureVerificationTarget:(id)self
     action:(SEL)@selector(onSignatureVerification:)

void removeSignatureVerificationTarget:(id)self
```

SignatureCapture Callback

The Payment Device SDK fires a SignatureCapture event during a transaction when a digital signature is required. Call continueSignatureCapture() to pass in digital signature data and continue the transaction.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addSignatureCaptureListener(
    ISignatureCaptureListener listener
)
void removeSignatureCaptureListener(
    ISignatureCaptureListener listener
)
```

iOS

```
void addSignatureCaptureTarget:(id)self
     action:(SEL)@selector(onSignatureCapture:)
void removeSignatureCaptureTarget:(id)self
```

Surcharge Callback

The Payment Device SDK fires a Surcharge event carrying the surcharge limits returned by the gateway. The integrating application responds by calling continueSurcharge(). A surcharge observer must be registered before startTransaction() is called.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addSurchargeListener(
    ISurchargeListener listener
)
void removeSurchargeListener(
    ISurchargeListener listener
)
```

iOS

```
void addSurchargeTarget:(id)self
     action:(SEL)@selector(onSurcharge:)
void removeSurchargeTarget:(id)self
```

The callback provides the maximum surcharge amount the merchant may apply (in minor units), the maximum percentage, the surcharge currency, and whether an operator PIN is required.If a surcharge cannot be applied (for example an ineligible card or a gateway communication failure) the maximum values are returned as 0 and an error such as ServerCommunicationError is included; in this case respond with Result set to FALSE.

ApplicationSelection Callback

The Payment Device SDK fires an applicationSelection event during a transaction when a customer is required to select the card application they wish to use for the transaction. Call continueApplicationSelection to pass the customers selection and continue the transaction.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addApplicationSelectionListener(
    IApplicationSelectionListener listener
)
void removeApplicationSelectionListener(
    IApplicationSelectionListener listener
)
```

iOS

```
void addCardApplicationSelectionTarget:(id)self
     action:(SEL)@selector(onCardApplicationSelection:)
void removeCardApplicationSelectionTarget:(id)self
```

UserNotification Callback

The Payment Device SDK fires a userNotification callback when a prompt is required to be shown to the customer. No integration call to the SDK is required to continue the transaction. The transaction will continue as normal.

Methods to add or remove callback delegate in ChipDnaMobile

Android

```
void addUserNotificationListener(
    IUserNotificationListener listener
)
void removeUserNotificationListener(
    IUserNotificationListener listener
)
```

iOS

```
void addUserNotificationTarget:(id)self
     action:(SEL)@selector(onUserNotification:)
void removeUserNotificationTarget:(id)self
```

11. Platform Specific Requirements

The SDK is available for iOS and Android platforms. There are some specific requirements for each platform and these are detailed below.

SQLCipher

SQLCipher is the SQLite database extension. It means that the default SQLite libraries provided by iOS APIs cannot be used in the same application, as this will cause unexpected behaviour. However, the provided SQLCipher libraries can be used by an integrating application for database access. The Payment Device SDK requires a password during initialisation of its own database but an empty password can be used for an unencrypted database used within the integrating application.

SQLCipher is required for the SDK to compile and run. The iOS SDK has the SQLcipher library and dependencies included in the release package, while in Android this must be included via Gradle by placing the following line in the applications dependencies:

compile 'net.zetetic:sqlcipher-android:4.7.2@aar'

ID TECH VP3350 SDK

Unlike other supported PIN pads the ID TECH VP3350 requires the ID TECH SDK to be added to the project in order for it to function. The ChipDNA Mobile SDK is supplied with the supported version of the ID TECH SDK.

In the Android release this can be found at libs/Universal_SDK_1.00.180_os.jar.

In the iOS release this can be found at ChipDnaMobile/IDTech.xcframework.

iOS also requires the inclusion of the ID TECH bundle which can be found at ChipDnaMobile/IDTech.bundle.To link this to your project add the bundle under Copy Bundle resources.



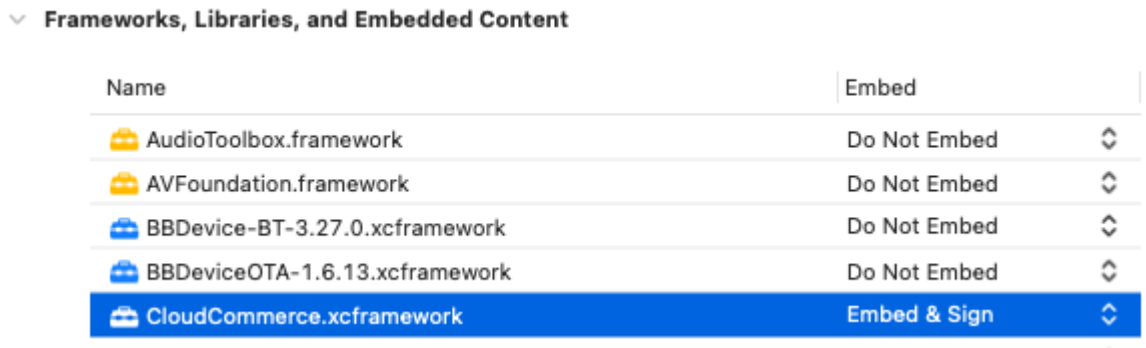
BBPOS SDK

Similarly to ID TECH VP3350 the BBPOS CHB30 requires the BBPOS SDKs to be added to the project in order for it to function. The ChipDNA Mobile SDK is supplied with the supported version of the BBPOS SDKs.

In the Android release this can be found at libs/bbdevice-android-3.29.1.jar and libs/bbdeviceota-android-1.6.28.jar.

In the iOS release this can be found at ChipDnaMobile/BBDeviceOTA-1.6.13.xcframework and ChipDnaMobile/BBDevice-BT-3.27.0.xcframework.

iOS



The minimum supported iOS version is 17.4.

A list of protocol strings must be presented by the application to enable the Payment Device SDK to communicate with an individual PIN pad. The following string must be added to the UISupportedExternalAccessoryProtocols property of an application's Info.plist file:

- com.miura.shuttle - To use the Miura Shuttle.
- com.idtechproducts.neo - To use the ID TECH VP3350.

Only add the protocol strings of the devices you will be supporting otherwise your application may get rejected by Apple during the App Store review.



BBPOS Chipper integrations do not require submission of an MFi Product Plan ID therefore there is no accessory protocol entry to be added to the Info.plist for these devices.

The “external-accessory” string must be added to the UIBackgroundModes property of an application’s Info.plist file.

The libraries CoreLocation, CoreData, ExternalAccessory, MessageUI, SystemConfiguration, Security, AVFoundation, MediaPlayer, CoreAudio, AudioToolbox and CoreBluetooth should be added to the application target's linked libraries as they are used by the SDK.

Libraries libsqlite3.dylib or libsqlite3.0.dylib, if present, should be removed from the linked libraries, as the database library provided with the SQLCipher libraries will be used instead.

Build settings should be changed to include Other C Flags:

• -DSQLITE_HAS_CODEC,

And Other Linker Flags:

• -ObjC

• -lz

• -lstdc++

Swift

To use the Payment Device SDK in a Swift project add the SDK files to your project in the usual way. Then create a new header file to use as a bridge header by going ‘File >> New >> File’ and selecting to create a new ‘.h’ file. In the new header file add import statements for all ‘.h’ files in the SDK.

Once the files are imported in the bridge header, select your project file from the navigation menu in Xcode. Under ‘Build Settings’ change the viewing option from ‘Basic’ to ‘All’ and scroll down to the ‘Swift Compiler – Code Generation’ section. Then set the ‘Objective-C Bridging Header’ value to the path of your newly created bridge header file.

For more information on using Objective-C code in a swift project please refer to the Apple documentation.

To ensure compatibility and avoid Swift compile-time errors, please use Xcode version 16.4.x or higher when developing your integration.

Android

The minimum supported Android version is 12 (SDK version 31).

Android has been designed such that no application, by default, has permission to perform any operations that would adversely affect the operating system or the user. Therefore, to utilise the complete API the following permissions must be granted:

- READ_PHONE_STATE - allows read only access to the phone state and is used to obtain a unique identifier for the mobile device required for a configuration update and to read the IMSI from the SIM to verify the Home Network Identity.
- BLUETOOTH and BLUETOOTH_ADMIN - allows applications to connect to Bluetooth devices.
- BLUETOOTH_SCAN and BLUETOOTH_CONNECT (SDK versions 31 and above) - allows applications to connect to Bluetooth devices.
- ACCESS_COARSE_LOCATION (SDK versions 23 and above) and ACCESS_FINE_LOCATION (SDK versions 30 and above) - allows applications to scan for and connect to Bluetooth Low Energy devices.
- INTERNET - allows applications to open network sockets.
- SCHEDULE_EXACT_ALARM (SDK versions 31 and above) - allows an application to schedule alarms that go off at an exact time.
- android.hardware.telephony feature - should be included and set to false, prevents the devices without telephony from being filtered out on Google Play.
- android.hardware.usb feature - should be set in Applications planning to use a USB connection to prevent the application being shown to devices without USB on the Google Play store.

The Android SDK also requires the following dependencies to be linked in order to compile and run. You can do this by adding the following to the application dependencies block in the build.gradle file.

```
implementation 'com.google.android.play:integrity:1.1.0'
implementation 'com.squareup.retrofit2:adapter-rxjava3:2.9.0'
implementation 'com.google.android.gms:play-services-location:21.0.1'
implementation 'com.squareup.okhttp3:logging-interceptor:4.12.0'
implementation 'io.reactivex.rxjava3:rxjava:3.0.0'
implementation 'io.reactivex.rxjava3:rxandroid:3.0.0'
implementation 'com.squareup.retrofit2:converter-scalars:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.google.code.gson:gson:2.13.2'
implementation 'commons-codec:commons-codec:1.11'
implementation 'com.squareup.okhttp3:okhttp-urlconnection:4.12.0'
implementation 'androidx.security:security-crypto:1.1.0-alpha06'
implementation 'com.google.android.gms:play-services-safetynet:17.0.0'
implementation 'org.slf4j:slf4j-api:2.0.18'
implementation 'ch.qos.logback:logback-classic:1.5.32'
implementation fileTree(include: ['*.jar'], dir: 'libs')
implementation 'androidx.core:core-ktx:1.8.0'
implementation 'org.jetbrains.kotlin:kotlin-stdlib:1.6.0'
implementation 'net.zetetic:sqlcipher-android:4.7.2@aar'
implementation 'androidx.sqlite:sqlite:2.1.0'
implementation 'com.jakewharton.timber:timber:4.7.1'
```

Applications that package the full build flavour must also exclude META-INF/INDEX.LIST from resource merging, as logback-classic and logback-core both provide that file:

```
packaging { resources { excludes += 'META-INF/INDEX.LIST' } }
```

More information about adding kotlin to existing projects can be found in the Android documentation here: https://developer.android.com/kotlin/add-kotlin

Obfuscation



The Payment Device SDK for Android is already obfuscated. Passing the binaries through secondary obfuscation can result in incorrect operation of the SDK including loss of data.

Below is an example ProGuard configuration for excluding the Payment Device SDK binaries from obfuscation.

```
-keep class net.sqlcipher.** { *; }
-keep class net.sqlcipher.database.* { *; }
-libraryjars /libs/ChipDnaMobile.jar
-libraryjars /libs/CardEaseXMLClient.jar
-libraryjars /libs/Universal_SDK_1.00.180_os.jar
-libraryjars /libs/bbdevice-android-3.29.1.jar
-libraryjars /libs/bbdeviceota-android-1.6.28.jar
-libraryjars /CloudCommerceSDK/cloud-commerce-sdk-release-6.9.0.aar
-keep class com.creditcall.** {
*;
}
```

12. Tap To Mobile

iOS

In order to use Tap To Mobile, you will also need to link against the CloudCommerce.xcframework

When adding the CloudCommerce.xcframework do so from your application target under the general section in “Frameworks Libraries and Embedded Content” and ensure that the Embed drop down is set to “Embed and Sign.”



The minimum supported iOS version depends on the SDK flavour: 18.5 for the Full flavour and 17.4 for the Lite flavour. See iOS Flavours.

Additional Permission Requirements

During configuration the Payment Device SDK will perform a suite of security checks, as part of this it will require the integrating application to be granted location permissions. The recommended level of permission is CLAuthorizationStatus.authorizedWhenInUse.

Apple has extensive documentation on the iOS CLLocationManager available here.

Updating Info.plist

PRODUCT_TEAM_IDENTIFIER

To properly attest the security of the application when using Tap To Mobile the team identifier is required to be added to the application info.plist

Permission usage descriptions

When requesting location permissions iOS will look for the following within the applications info.plist

- Privacy - NFC Scan Usage Description
- Privacy - Location When In Use Usage Description
- Privacy - Location Always and When In Use Usage Description

Example

Key	Type	Value
PRODUCT_TEAM_IDENTIFIER	String	\<*your team identifier*>
Privacy - NFC Scan Usage Description	String	This application needs to read NFC card & tags in order to accept payments.
Privacy - Location When In Use Usage Description	String	This application needs to check that you are operating the app from a permitted location.
Privacy - Location Always and When In Use Usage Description	String	We'd like to access your location to store it with each transaction.

Apple Sandbox Account

To develop an application utilizing Apple's Tap to Pay technology, the device must be signed into a developer sandbox account.

These can easily be created from App Store Connect under “Users And Access > Sandbox”

Entitlements

Apple Tap to Pay requires additional entitlements, which must be requested by the Apple Developer Account Holder. These entitlements are divided into two types:

1. **Developer Entitlements:** Enables application development on devices registered under the Apple Developer account.
2. **Publishing Entitlements:** Used when the application is ready for release on the Apple App Store.

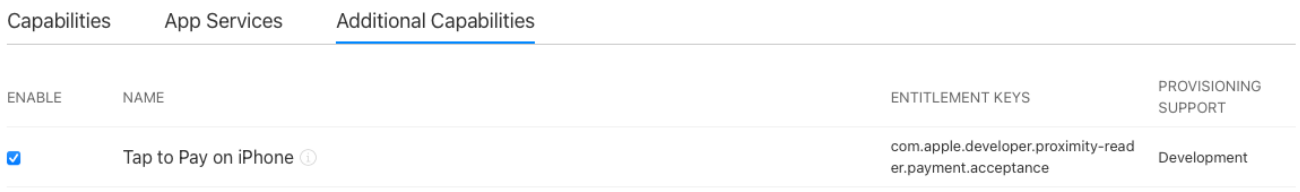
The form for requesting entitlements can be found here.

Capabilities

Once you've received your developer entitlements you'll need to add the following capabilities to your application's Identifier:

- App Attest
- NFC Tag Reading
- Tap To Pay on iPhone

Tap To Pay on iPhone will appear once your development entitlements have been approved by Apple at which point it'll appear under "Additional Capabilities" when configuring your application's Identifier.



Once the application's Identifier is configured you must create a new provisioning profile linked to it and link to it in your xCode project.

Within your application's xCode project you will then need to update your entitlements file with the following:

Key	Type	Value
App Attest Environment	String	developement
com.apple.developer.proximity-reader.payment.acceptance	Boolean	YES
Near Field Communication Tag Reader Session Formats	Array	1 Item
Item 0 (Near Field Communication Tag Reader Session Formats)	*String*	*Tag-Specific Data Protocol (TAG)*

Full details on how to setup and integrate into Tap To Mobile can be found in the Payment Device SDK for iOS Tap To Mobile Supplement provided with the Payment SDK Release.

Android

Implementing TapToMobile requires adding the Cloud Commerce SDK to your Android project. We include two AARs in our release package:

- Test AAR (cloud-commerce-sdk-debug): built as debuggable and preconfigured to use only sandbox endpoints. You get verbose logs and can't accidentally hit a live endpoint.
- Production AAR (cloud-commerce-sdk-release): non-debuggable and minimal logging and locked to live endpoints under full PCI-compliant security controls.

Keeping them separate protects against accidental use of production resources in development and ensures your production build remains lean, secure, and compliant.

In the Android release these can be found at:

- Test SDK libs/full/cloud-commerce-sdk-debug-6.9.0.aar
- Production SDK libs/full/cloud-commerce-sdk-release-6.9.0.aar

To integrate the Cloud Commerce SDK a new module is required. This module needs to contain the cloud commerce aar files and a build.gradle file. The gradle file wants to contain the following (depending on the aar you wish to implement):

configurations.maybeCreate("default")

artifacts.add("default",file("./cloud-commerce-sdk-debug-6.9.0.aar"))

The settings.gradle file of the integrating application will also want to contain the lines, replacing {ModuleName} and {PathFromRepositoryRoot} with values of the new module containing the cloud-commerce-sdk :

```
include ':{ModuleName}'
project(':{ModuleName}).projectDir=new File('{PathFromRepositoryRoot}')
```

The project must be implemented in the applications app level build.gradle file:

implementation project (":CloudCommerceSDK")

Additional Requirements

To initialize the Payment Device SDK to allow TapToMobile, it is required that the integrating application must contain an application class that extends ChipDnaApplication. This is due to

MPoC requirements for the TapToPay solution. APIs exposed by Cloud Commerce SDK can only be consumed through their own backend environment which our ChipDnaApplication extends. It manages the initialization of the SDK instance to enable API consumption by the app module. In addition to this, it also performs security checks in the background and disables access to API further in case of any vulnerability detected.

The success of the SDK initialisation can be observed by the integrating app through the following override methods:

```
void onSDKInitializationSuccess()

void onSDKInitializationFailed(
errorMessage: String?
)
```

The application class that extends ChipDnaApplication must also be included in the AndroidManifest.

```
<application
android:name=".ApplicationClassName"
```

Additional Permissions

During configuration the Payment Device SDK will perform a suite of security checks, including requiring the integrating application to be granted location permissions. On top of the standard permission requirements, the recommended level of permission is:

- android.permission.RECORD_AUDIO - for verifying if any recording is ongoing when entering a PIN
- android.permission.HIDE_OVERLAY_WINDOWS - Allows an SDK to prevent and verify if any non-system overlay windows are being drawn on top of the app
- android.permission.NFC - Allows an SDK to perform I/O operations over NFC.
- android.permission.VIBRATE - Required for vibrating the application on card detection.

Updating Dependencies

The following dependencies may need to be updated to the versions shown below or higher:

- com.android.tools.build:gradle:8.7.2
- JavaVersion.VERSION_17

Attestation Requirements

In order to ensure security and MPoc compliance, we require the following information to form our attestation:

- ***App Package name*** **- Package name of the integrating app.**
- **Key Store Hash** - This is a SHA-256 hash value from the keystore certificate (public key).
- **Play Integrity Keys** - These are response encryption keys used to secure the communication between your Android application and the Google Play services.

You will require a **PEM file** from us to obtain your Play Integrity keys. You will also require a **Google Play Console** and **Google Play Project.**



Full details on how to setup and integrate into Tap To Mobile can be found in the Payment Device SDK for Android Tap To Mobile Supplement provided with the Payment SDK Release

Display Requirements

When the integrating application prompts for card entry, the activity passed into our Activity Callback must comply with EMVCo Contactless and Card Brand requirements. These requirements can be found in the Payment Device SDK for Android Tap To Mobile Supplement provided with the Payment SDK Release.

13. Device Information

ID TECH VP3350 BLE



Please be aware that if an ID TECH VP3350 device has a low battery, it may fail to connect to the SDK. If connection issues arise and the first LED is blinking red, it is advisable to charge the device.

Power Button Operation

When the device is turned off press the power button to turn on the device. The device will automatically enter Bluetooth pairing mode.

When the device is not connected to a power source the device will enter a sleep mode and disconnect. Pressing the power button will wake up the device and a call to connectAndConfigure() must then be made to re-establish a connection.

LED Indicators



- All LEDs off - Device powered off or in standby mode.
- First LED Blinking Green - Device Powered on.
- First LED Blinking Red - Device Low battery.
- All LED flash green once - Bluetooth link established.
- All LEDS flash red once - Bluetooth link dropped.
- First LED green - Awaiting card entry.
- All LEDS flashing red - Device in a tampered state.
- All LED flashing yellow - Device deactivated.

More information can be found in our VP3350 Quick Start guides here:https://docs.nmi.dev/docs/vp3350-quick-start

BBPOS Chipper 3X BT



The following information pertains to the operation of the BBPOS Chipper 3X BT device.

Power Button Operations

Power On

To power on a BBPOS Chipper 3X BT press the power button once. The device will Beep twice and the LED light will turn on to indicate the device has power up.

Power Off

To power off a BBPOS Chipper 3X BT press and hold the power button for 4 seconds.

Clear Bluetooth Pairing Keys

To clear the Bluetooth pairing keys from the device. Press the power button 5 times, then press and hold for 4 seconds.

Clearing the Bluetooth Pairing keys on the device will require the device to be re-paired. On Android devices this will need to be done via the Android settings menu.

More information can be found in our Chipper Quick Start guides here:https://docs.nmi.dev/docs/chb30-quick-start-guide

LED Indicator



- All LEDs off - Device powered off.
- Red LED on - Device starting up / Charging battery.
- Red LED off - Fully charged.
- Red LED flashing - Low battery / Critical low battery.
- Blue LED on - Bluetooth is connected.
- Blue LED flashing - Standby mode / waiting for connect bluetooth or bluetooth is disconnected.

Datecs Bluepad 50

Device Menu

To access the devices menu press the Yellow <- button 3 times in quick succession.

Pairing Mode

To enter pairing mode Red (C) -> Yellow (<-) -> Green (OK)In quick succession.

Miura

Power On

To power on the device hold down the ‘X’ (red) button until the device beeps and shows the Miura Systems logo.

Power Off

To power off the device hold down the ‘X’ (red) button until the device displays Shutting Down prompt.

Device Menu

To access the device menu press the ‘←’ (yellow) button while the device is disconnected showing its home screen.

Enter pairing mode

To enter pairing mode hold down the ` ✓ ’ (green) button until the bluetooth indicator on screen begins to flash.

Wifi Configuration Menu

As well as via the Device Menu the Wifi Configuration menu can also be access by holding the ` ✓ ’ (green) button until the menu appears.

Ingenico Axium



Ingenico Axium devices operate in a similar manner to mobile Android devices.

Wifi Configuration



Ingenico Axium devices must have an internet connection for transactions to be processed.

To connect the device to a wifi network, either pull down on the top task bar and select the wifi symbol, or open the Settings application and select Network and Internet.

When a password is required, enter **350000. Once the password is entered, the device can be configured to connect to a Wi-Fi network.**



Ingenico Axium devices can also be connected to the internet via Ethernet or SIM card.

14. Connection Requirements

USB / Lightning

The Payment Device SDK for Android has the ability to connect to devices over USB to compatible Android devices.

ID TECH VP3350 Android

1. Plug the ID TECH USB-C PIN pad into the Android device.
2. A pop-up will display asking for permission to be given to the USB Manager to access the device.
3. Grant access to the USB Manager.

Supported Devices

Part Number	Connection Type	Supported SDK Version
IDMR-NUR93	USB-C Male	3.13+
IDMR-NUF93	USB-C Female	3.16.5+

ID TECH VP3350 IOS (Lightning and USB-C)

1. Ensure you have correctly added the required UISupportedExternalAccessoryProtocols string to the projects info.plist.
2. Plug in the ID TECH Lightning or USB-C PIN pad into the iOS device.
3. Search for the PIN pad using the getAvailablePinPads() method.

Supported Devices

Part Number	Connection Type	Supported SDK Version
IDMR-NUR93	USB-C Male	3.13+
IDMR-NLF93	Lightning	3.13+

When switching between a lightning ID TECH VP3350 and a Bluetooth ID TECH VP3350 you will need to unplug the lighting device before a connection can be made to the Bluetooth device.

USB devices now behave identically to Bluetooth devices. All that is required is setting the PIN pad name from getAvailablePinPads() along with a connection type.

Bluetooth

Miura

1. Turn the PIN pad on and wait until MIURA SYSTEMS is displayed along with a Bluetooth and battery indicator. Press and hold the green tick button until the Bluetooth indicator is flashing to show it is now discoverable. (If the PIN pad has never been paired the Bluetooth indicator will already be flashing).
2. On the mobile device select the PIN pad from the list of detected Bluetooth devices. The mobile device should display a prompt containing a passkey and the same passkey should be displayed on the PIN pad.
3. If the two passkeys are the same, press ‘Pair’ on the mobile device and the green tick button on the PIN pad. Once pairing is complete this should be reported on the mobile device and the PIN pad should return to its standby screen.

BBPOS Chipper 2X BT

1. Turn the payment device on by pressing the power button on the side. Wait until the LED indicator continuously flashes blue.
2. The BBPOS Chipper PIN pads are supported with Bluetooth Low Energy (BLE) connection instead of Bluetooth Classic connection. The mobile device and the BBPOS Chipper do not need to be paired before connecting.
3. The device will now appear in the list of available payment devices within the SDK.

Pairing with the BBPOS Chipper 2X BT on iOS will prevent the mobile and PIN pad from being able to communicate.

BBPOS Chipper 3X BT

1. Turn the payment device on by pressing the power button on the side. Wait until the LED indicator continuously flashes blue.
2. The BBPOS Chipper PIN pads are supported with Bluetooth Low Energy (BLE) connection instead of Bluetooth Classic connection. The mobile device and the BBPOS Chipper do not need to be paired before connecting.
3. The device will now appear in the list of available payment devices within the SDK.
4. The mobile device will require a pin to be entered upon connection to the PIN pad, this number can be found on the back of the Chipper.



Pairing with the BBPOS Chipper 3X BT on iOS will prevent the mobile and PIN pad from being able to communicate.

ID TECH VP3350 Android

1. Turn the payment device on by pressing the power button on the side.
2. On the mobile device, within the mobile device's settings menu, select the PIN pad from the list of detected Bluetooth devices. The mobile device should display a prompt containing a passkey, enter "123456" and press ok.
3. The device will now appear in the list of available payment devices within the SDK when the method getAvailablePinPads() is called.

ID TECH VP3350 IOS

1. Turn the payment device on by pressing the power button on the side.
2. Initialise the ChipDNA Mobile SDK and search for the payment device using the getAvailablePinPads() method.
3. Pass in the selected payment device into the ChipDNA Mobile SDK's connectAndConfigure() method.
4. When prompted enter the passkey "123456" and press ok.

Ingenico Axium

1. Turn the payment device on, making sure bluetooth is enabled.

2. On the mobile device, within the mobile device's settings menu, navigate to Connected Devices → pair new device and select the payment device. Both the mobile device and payment device should display a pairing prompt, with matching Bluetooth pairing codes. Select pair on each device.
3. The payment device will now appear in the list of available payment devices within the SDK when the method getAvailablePinPads() is called.

Wi-Fi

The SDK has the ability to connect to Wi-Fi capable devices. In order to do this the device must first be configured according to the manufacturer's instructions to connect to a Wi-Fi connection and an IP address and port number obtained. It's recommended that the PIN pad is configured to use a static IP and port number so that the IP address of the PIN pad doesn't change. If the IP address were to change then the SDK would require re-configuration before a connection could be re-established to the PIN pad.

The SDK must then be configured by passing in the following parameter keys into the setProperties() method:

- PinPadIpAddress – IP address configured on the PIN pad.
- PinPadPort – port number configured on the PIN pad.
- PinPadConnectionType – With parameter value TcpIpConnectionType.
- PinPadName – A PIN pad name in the same style as the Bluetooth name of the PIN pad. For example a Miura device with a serial number 020-1234567 would have the PIN pad name "Miura 567".

Appendix 1 - Supported PIN Pads

The Payment Device SDK supports a variety of different PIN pads. Table 3 details the currently supported PIN pads.

Please contact your payment gateway provider to purchase an approved PIN pad. PIN pads obtained outside of approved channels cannot be supported.

Table 3 - Supported PIN pads.

Manufacturer	Device	Region	EMV Contact (Chip)	EMV Contactless (Chip)
BBPOS	Chipper 2X BT*	US	Ready	Ready
BBPOS	Chipper 3X BT**	US	Ready	Ready
ID TECH	VP3350	US	Ready	Ready
Ingenico	Axium***	US	Ready	Ready

*BBPOS Chipper 2X BT models supported are CHB20, CHB22 and CHB29. **BBPOS Chipper 3X BT models supported are CHB30.

***Ingenico Axium models supported are DX4000, DX8000, EX4000, EX6000, EX8000 and RX5000.

Ingenico Axium devices are only supported when the Android payment device SDK is set up in a Smart POS integration.

Appendix 2 - TMS Properties

The following are the TMS properties which can be changed for the integration, as required. To change these properties, please contact Payment Device SDK support. The defaults stated are for the test platform. The values for the live platform should be agreed with your payment gateway provider.

Property	Description	Default Value
Gratuity	Enables gratuity.	Disabled.
Merchant Tipping	Enables merchant tipping, where the tip is passed in TipAmount on startTransaction(). Ask support to enable it; in TMS it is the third value of the TippingSupported setting. It is not reported by getStatus. Required for TipAmount on the VP3350 and Chipper PIN pads (Android) and on Tap To Mobile.	Disabled.
Standalone Refunds	Enables standalone refunds.	Enabled.
Linked Refunds	Enables linked refunds.	Enabled.
Digital Signature Capture	Enables the digital signature capture.	Disabled.

Appendix 3 - Receipting

Receipts must be issued after the transaction result is returned. The Payment Device SDK can be used to issue receipts for the last sale or refund transaction completed using either:

- SMS
- Email

Receipts sent using SMS or email can be sent either from the mobile device using the returned receipt data or the central servers if enabled via TMS settings.

There are cost implications for sending using the central server so please contact support for further information.

The receipt content has been certified by the schemes and acquirers and its content cannot be modified.

SMS messages sent from the central servers require the international dialling code. If the international dialling code is not provided with the phone number, UK values will be used as default – “+44” as international dialling code and “0” as trunk prefix, which will be removed from the phone number when international dialling code is added. Different default values can be configured for the terminal using a TMS setting, for example US numbers will use “+1” as international dialling code followed by 3-digit area code prefix.

Appendix 4 - App Submission

Before Submission

Before your application is released, TMS settings for that application identifier should be confirmed. This includes settings like the receipt source, whether the digital signature is supported, etc.

IOS App Store Submission

The applications submitted to AppStore go through the review process. There are several things to be aware of to ensure that the application is approved quickly without rejections.

Protocol strings should be reviewed before submitting an application. Only the supported accessories' protocol strings should be declared. Review notes should include the MFi Product Plan ID for the supported accessory. The manufacturer of the accessory will also need to register your application as using that specific accessory. You should contact the manufacturer of the accessory to arrange this.

Either a user guide or a video demonstrating how to use the application should be provided in the review notes. It is not currently possible to upload these resources directly to the iTunes Connect, as such, an external link should be provided, with the credentials to access the resources, if needed.

Review notes should also include test account credentials to ensure that the application can be tested.

PIN pads do not have to be provided for testing. They are available for Apple through the MFi program from the manufacturers.

Google Play Store Submission

An application can be submitted as normal through Google Play. However, in submitting an application you confirm that you have complied with the relevant export compliance laws. As such, the relevant forms should still be filled in.

Export Compliance

The Payment Device SDKs for both iOS and Android use SQLCipher. As encryption is used, the application is subject to export compliance with iOS AppStore and Android Google Play. This section explains how to obtain an encryption registration and how to submit an application which uses encryption.

If the application will only be distributed in the USA and/or Canada, the encryption registration is not required. However, during an iOS application upload, it should still be declared that an application uses encryption. Application metadata should be set correctly to only USA and/or Canada to ensure that the registration does not need to be submitted.

If the application is distributed in any other country, the encryption registration is required. This is the case even if the application is not distributed in the USA. This is because the app stores are based in the USA.

If the application is distributed in France, additional registration for export compliance in France might be required. Registration is not required for the banking applications.

According to the French regulation (décret 2007/663, annexe 1, cat3), banking applications are not subject to declaration. (Last Accessed Oct 2018)

https://www.legifrance.gouv.fr/codes/article_lc/LEGIARTI000006428332/2024-02-23/

Please use this as a guide only. If your application has any additional functionality it might influence the export registration or the approval from the AppStore. It is your responsibility to ensure that your application complies with all regulations in all regions where your application will be distributed.

Please also refer to the FAQ on iTunes Connect or Google Play Store support, as they have a section on export compliance as well.

Appendix 5 - Firewall Configuration

The Payment Device SDK must be able to communicate with CardEaseXML (https://live.cardeasexml.com) and CardEaseTMS (https://tms.cardeasexml.com) on the payment gateway platform. This may require changes to your firewall configuration.

The table below lists the IP addresses and port numbers for each external service used by the SDK.

Table 4 - IP addresses and port numbers for external services.

Service	Platform	IP addresses	Ports
CardEaseXML	Live	91.197.92.231 91.197.93.231 91.197.94.204 91.197.95.231 91.197.92.250 91.197.93.250 91.197.93.251 91.197.94.250 91.197.94.252 74.120.0.250 74.120.1.250 74.120.1.251 74.120.2.250 74.120.2.252	443
	Test	91.197.92.230 91.197.93.230 91.197.94.203 91.197.95.230	443
CardEaseTMS	Live	91.197.92.239 91.197.93.239 91.197.94.239 74.120.0.239 74.120.1.239 74.120.2.239	443
	Test	91.197.92.219 91.197.93.219 91.197.94.219 91.197.95.219	443
Smart POS	Live	193.163.48.126 35.195.97.84	443 7037
	Test	193.163.48.127 35.195.97.84	443 7037
Merchant Gateway	Live	104.192.33.49	443
	Test	104.192.33.188	443

Appendix 6 - Smart POS Supported API Methods

This table shows all API methods in the Payment Device SDK and shows their supported status with Smart POS integrations.

Table 5 - Supported Smart POS API Methods

API Method	Supported	Notes
initialize	Yes	Requires passing of IsSmartPosIntegration parameter key with value of TRUE
setProperties	Yes	-
connectAndConfigure	Yes	-
getAvailableCurrencies	Yes	Returns the merchant's boarded currencies (GBP and USD with an earlier Transaction Initiator UI)
getAvailablePinPads	Yes	-
getStatus	Yes	Requires connection to payment device for full status
getTransactionInformation	Yes	Requires connection to payment device
getMerchantData	No	-
requestTmsUpdate	Yes	-
disconnect	Yes	-
dispose	Yes	-
startTransaction	Yes	-
continueDeferredAuthorization	No	-
continueForceAcceptance	No	-
continuePartialApproval	Yes	-
continueSignatureVerification	Yes	-
continueSurcharge	No	-
continueVoiceReferral	No	-
continueSignatureCapture	No	Signature capture performed on payment device
continueApplicationSelection	No	-
confirmTransaction	Yes	Requires connection to payment device
voidTransaction	Yes	Requires connection to payment device
linkedRefundTransaction	Yes	Requires connection to payment device
getCardDetails	Yes	-
terminateTransaction	Yes	-

API Method	Supported	Notes
processReceipt	Yes	Printing only, with Transaction Initiator UI v1.4.0 or later. Email and SMS are not supported
setIdleMessage	No	-
deleteFailedOfflineRequest	No	-
retryFailedOfflineRequest	No	-
processOfflineRequest	Yes	-

Appendix 7 - Smart POS Supported Features

This table details the transaction features that are not supported on Smart POS integrations compared to Non-Smart POS integrations.

Table 6 - Supported Smart POS Features

Feature	Supported
Delay Online Processing	No
On Device Tipping	No
Customer Vault	Yes
Merchant Defined Fields	No
Purchase Order	No
Tax Amount	No
Credential On File	Yes
Dynamic tipping	No
One Way Card Tokens	Yes
Surcharging	No

Appendix 8 - Axium Device Modes

Development

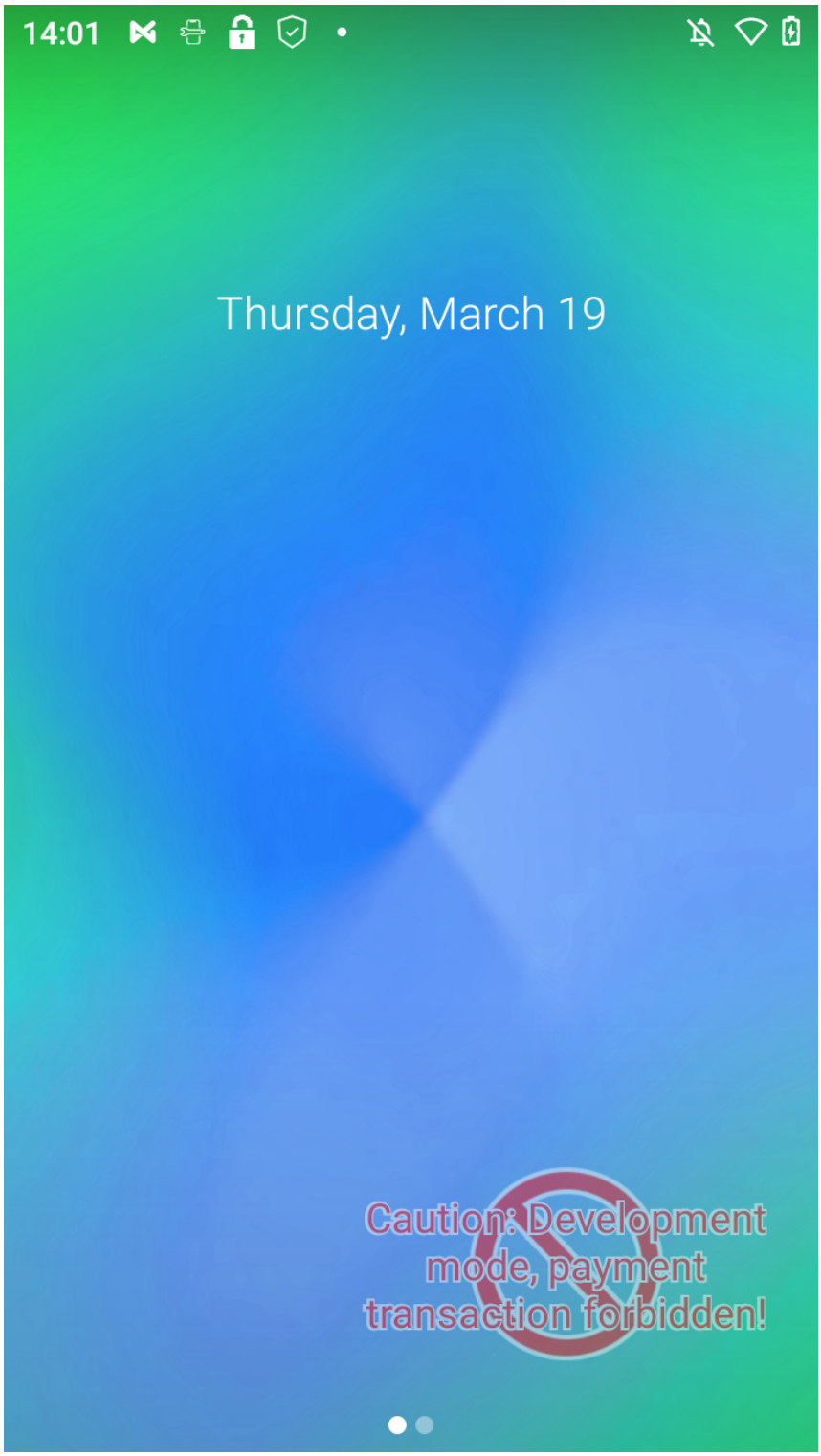
Development devices are injected with test keys and can only be used in the Test environment. They are intended for building and testing applications directly on the device.

These devices:

- Ignore application signature validation
- Can be connected to Android Studio
- Display a warning message on the home screen

Once a device has been set to Development mode, it cannot be reverted to Production mode without being returned to Ingenico.

You can tell development devices at a glance as they will flash a warning on the home screen



Production

Production devices are injected with production keys and can be used in both Test and Production environments. These are intended for standard usage where no on-device development is required.

These devices:

- Only run applications signed via Ingenico’s e-signing portal
- Do not appear in Android Studio
- Cannot be used for development