Payment Device SDK for Android

Axium Smart POS Supplement for Release 5.4.2 (Cassiopeia)

NMI
© NMI. All rights reserved.

Table of contents

1. Introduction 5
2. High Level Overview 5
3. Preparing for Development 7
   - Pre-requisites 8
     - Android Flavours 8
       - Lite 8
       - Full 8
   - Device Setup 9
     - Installing Apps 9
     - Wi-Fi Configuration 10
     - Bluetooth Configuration 11
       - Bluetooth Device Name 11
   - Supported Devices 11
     - Axium RX5000 12
       - Setting Up the Device 12
       - Restarting the Device 12
     - Axium EX4000, Axium EX6000 and Axium EX8000 12
       - Setting Up the Device 12
       - Charging the Device 12
       - Restarting the Device 12
     - Axium DX4000 and DX8000 13
       - Setting Up the Device 13
       - Restarting the Device 13
       - Charging the DX4000 Device 13
       - Charging the DX8000 Device 13
4. Smart POS API Updates 14
   - Initialize 14
     - Parameters Required for a Smart POS Initialize 14
   - Set Properties 15
   - Connect And Configure 15
   - Get Status 16
   - Get Available Currencies 17
   - Start Transaction 17
     - Parameters Required for a Smart POS Transaction 18
     - Parameters That Will Cause Errors for a Smart POS Transaction 18

New Parameters Available for a Smart POS Transaction 18

ParameterKeys.TransactionAuthPreference 19

ParameterKeys.OfflineTransactionUploadMode 19

ParameterKeys.MagstripeDebit 19

ParameterKeys.MagstripeSignature 19

ParameterKeys.PartialApproval 20

ParameterKeys.TipAmount 20

ParameterKeys.CardEntryTimeout 20

ParameterKeys.FeatureTokens 20

Process Receipt 20

Get Transaction Information 21

Methods Requiring Smart POS Device Connection 21

5. Events 22

Transaction Update Events 22

TransactionUpdate 22

New Error Codes 22

Connect And Configure 22

Process Receipt 23

Start Transaction 24

Transaction Finished 24

6. Offline Transaction Queue Processing 25

7. Device Disconnection During a Transaction 25

8. Retrieving Log Files 27

9. OTA Application Updates 27

Triggering Update via Axium Device 28

Triggering Update via ChipDNA Mobile 30

Appendix 1 - Supported API Methods 32

Appendix 2 - Smart POS Unsupported Features 33

Appendix 3 - Axium Data Usage 33

Smart POS Supplement

Android supplement for Payment Device SDK Release 5.4.2 (Cassiopeia).

Document History

Document Version	Software Version	Date (yyyy-mm-dd)	Summary of Changes
1.0.0	4.1.0	2025-09-17	Document Created
1.0.1	4.1.0	2025-10-02	Added information on how to change the bluetooth name of an Axium device. Add new error code.
2.0.0	5.0.0	2025-11-14	Updated version of live pilot. Added section on how to retrieve log files. Updated supported features. Added information on how disconnections are handled during a transaction. Updated information around device bluetooth name.
2.0.1	5.0.1	2026-01-09	Updated for PAN Key entry support
2.0.2	5.0.1	2026-01-28	Added warning that remote linked refunds are not supported on Axium devices.
2.0.3	5.1.0	2026-02-11	Support for linked refunds added on smart POS integrations. Refunds on smart POS integrations don't support auto confirm, they are always confirmed and so transaction status will indicate if a transaction needs to be confirmed.
2.0.4	5.2.0	2026-03-25	Update document version. Update to high level overview. Added section for OTA updates.
2.0.5	5.3.0	2026-05-01	Updated documentation version.
2.0.6	5.3.1	2026-06-19	Updated documentation version. Removed live pilot restriction around deferred auth preference flag.
2.0.7	5.4.0	2026-07-30	Updated documentation version. Added Credential on File first store support for SmartPOS integrations. Resolved issue whereby ConnectAndConfigure required an online connection to succeed.
2.0.8	5.4.1	2026-08-14	Updated documentation version. Added Appendix 3 detailing Axium data usage per merchant action.
2.0.9	5.4.2	2026-10-30	Updated documentation version. Added receipt printing. Added the admin settings (kiosk mode, notification panel and administrator passcode) to Connect And Configure and Get Status. Added the CardEntryTimeout parameter. Added FeatureTokens and FundingCardTokens. Currencies now come from the merchant's boarding.

1. Introduction

This supplement document contains additional information about how to set up Smart POS (Point Of Sale) functionality in the Payment Device SDK.

This document details any pre-requisites required, configuring your development environment, and how to prepare your application for release. It also goes through the Smart POS integration specific API changes that integrations will need to be aware of.



If you're intending to develop an application that runs on the Axium device you should use this document in conjunction with the Axium Development Guide Supplement.

2. High Level Overview

The component diagram shows the interactions between the components in a Smart POS integration with the Payment Device SDK.

The mobile device hosting the integrating application will require a Bluetooth connection to the Ingenico Axium device, to communicate with the NMI Android All In One (AAIO) service.

When integrating in All In One mode, connection to the NMI AAIO service operates over a local TCP/IP connection.

The AAIO service communicates with the Axium Payment Service to run transactions on the Ingenico Axium device, and requires an internet connection to the NMI Backend to process the payments.

The Axium Payment Service requires an internet connection to the Ingenico Hardware Estate Management (HEM), to monitor and update software on the Ingenico Axium device.

This differs from existing Payment Device SDK integrations, where the mobile device required the internet connection to process payments with the NMI Backend.

Diagram 1 – Components of Smart POS Local Semi-Integration

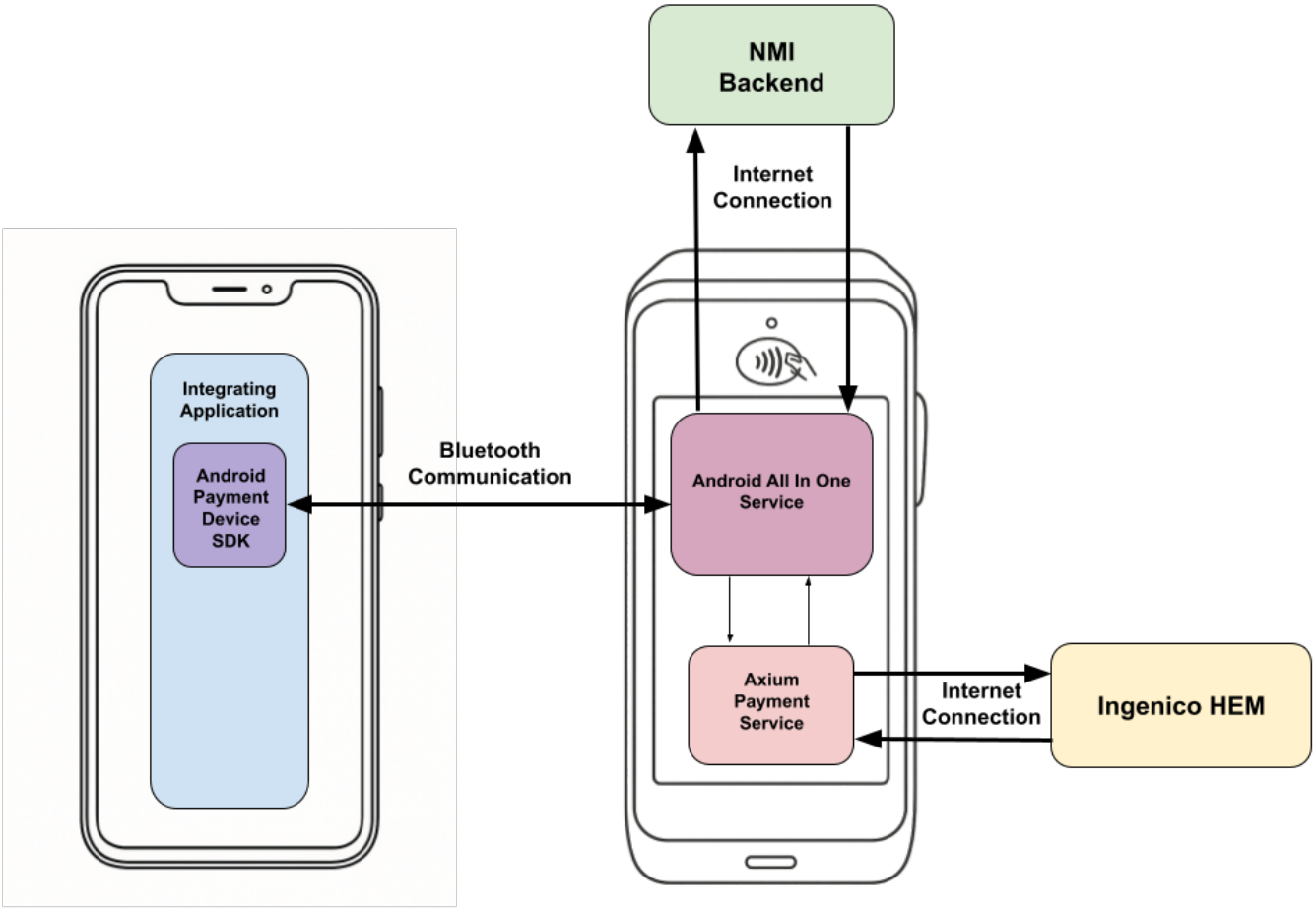
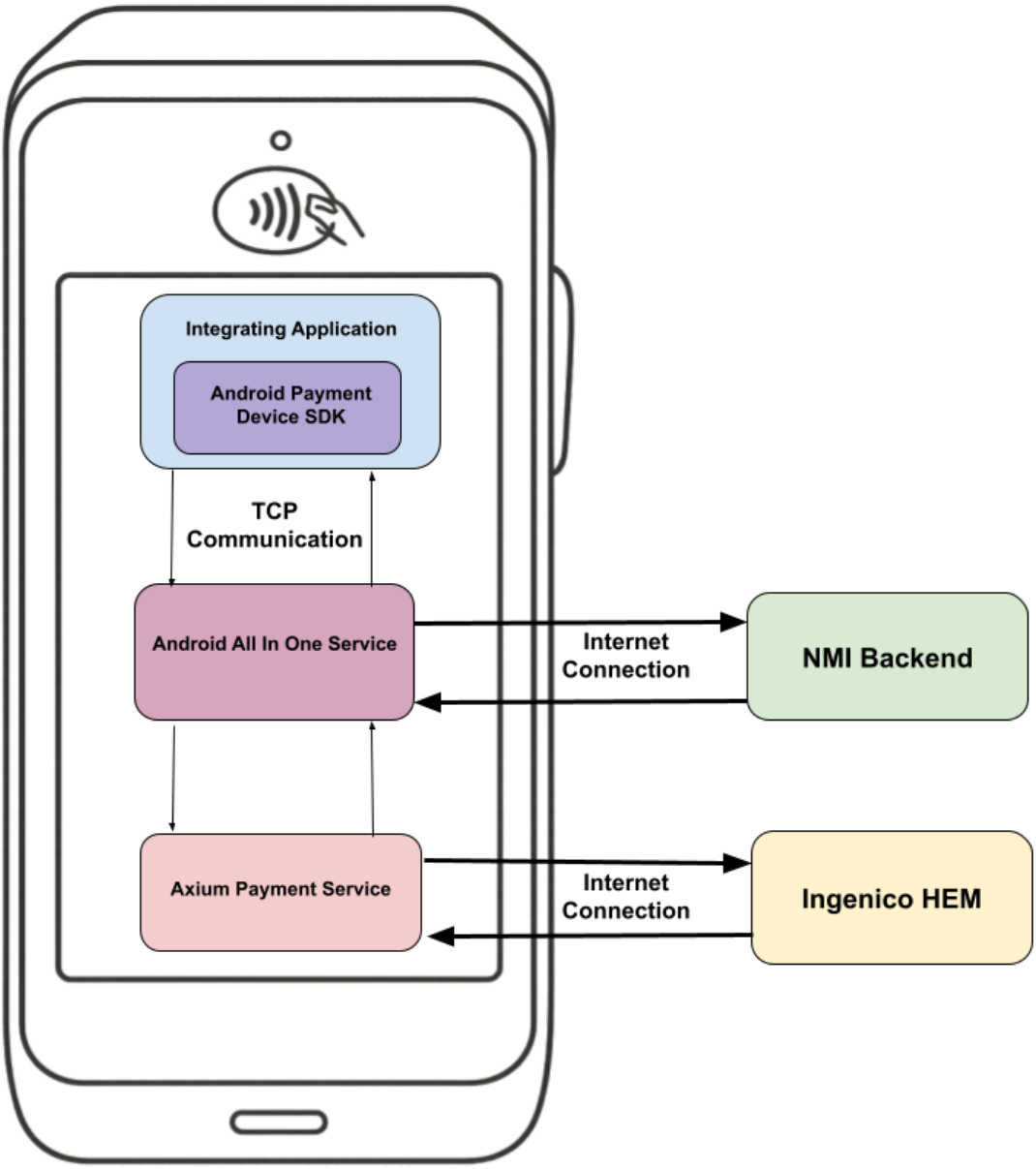


Diagram 2 – Components of Smart POS All In One Integration



3. Preparing for Development

If intending to integrate with the Axium device in All In One mode please also refer to the Axium Development Guide Supplement.

Pre-requisites

The SDK software is packaged and supplied as `.zip` archives, where the contents are described in Table 1.

Table 1 – Contents of NMI Payment Device 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.
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`



Do not include both SDK flavours in the same project. Choose the one that best fits your integration approach. When enabling Smart POS Integration, all other POS devices will be unavailable.

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`.



The Lite flavour is recommended for Smart POS integrations.

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.

Device Setup

Installing Apps

Your Axium device will come with all the applications and firmware needed to accept payments already installed.

The following apps should be installed on your device when you receive it; there may be some additional apps as well.

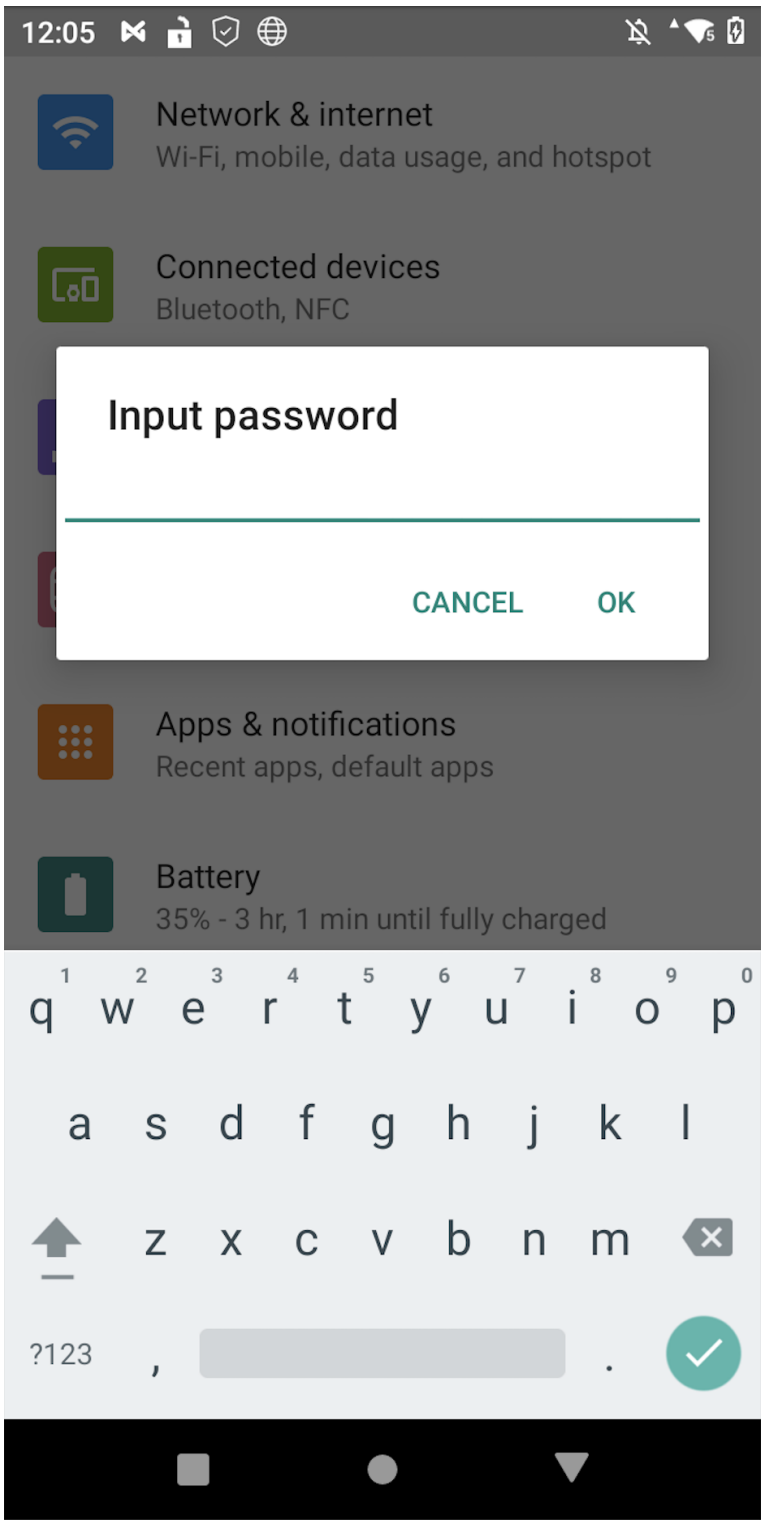
- AppStore
- AXIUM Retail Core
- ARC Comm Config
- AXIUM UX
- View Security Settings App
- TransactionInitiatorUI

If any of these are not present please contact integration support.



Ingenico Axium devices run on a security-hardened version of Android, so while they operate in a similar way to standard Android devices, additional security measures are enforced.

Wi-Fi Configuration



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

To connect the device to a Wi-Fi network, either pull down on the top task bar and select the Wi-Fi 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.

Bluetooth Configuration

To connect to the device via Bluetooth:

1. Turn the payment device on, and enable Bluetooth.

a. Navigate to the Settings app.

b. Select Connected devices.

c. Select Connection preferences.

d. Select Bluetooth.

e. Check the top On/Off bar below the title:

- If it is grayed out and displaying Off, tap the switch once to turn Bluetooth on.
- If it is already highlighted and displaying On, 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.

BLUETOOTH DEVICE NAME

When the Transaction Initiator UI is first installed on an Axium device, it will set the Bluetooth name of the device to the following format:

AXIUM-{model number}-{serial number}, e.g. **AXIUM-DX6000-235GKD420151**.

Supported Devices

The following Axium Smart POS devices are supported.

Axium RX5000

SETTING UP THE DEVICE

1. **Connect the HDMI cable** to the port on the back of the device.
2. **Use the split cable**:
   - Plug the **power pin** into a power socket.
   - *(The USB connector is not required for normal use, as connections are made via Bluetooth.)*

RESTARTING THE DEVICE

1. Press and hold the **Green "OK" button** and the **dot (.) button** at the same time.
2. Two tiles will appear at the top of the screen:
   - **Services** (not normally needed)
   - **Restart** (select this option)

Axium EX4000, Axium EX6000 and Axium EX8000

SETTING UP THE DEVICE

1. Press the **orange power button** on the right side of the device to turn it on.

CHARGING THE DEVICE

1. A **USB-C port** is located on the left side of the device for charging.
2. Connect the charging cable to this port until the device is fully charged.

RESTARTING THE DEVICE

1. Press and hold the **power button** for a few seconds.
2. A menu will appear with the following options:
   - **Power off**
   - **Restart** (select this option)
   - **Screenshot**
   - **Services** (not normally needed)



Ingenico Axium EX4000 devices cannot perform magstripe transactions.

Axium DX4000 and DX8000

SETTING UP THE DEVICE

1. Press the **power button** on the left side of the device to turn it on.

RESTARTING THE DEVICE

1. Press and hold the **power button** for a few seconds.
2. A menu will appear with the following options:
   - **Power off**
   - **Restart** (select this option)
   - **Screenshot**
   - **Services** (not normally needed)

CHARGING THE DX4000 DEVICE

1. **Remove the back cover**:
   - Push the tab located on the back of the device upwards.
   - Lift off the panel to reveal the battery and USB-C charging port.
2. **Connect the charging cable**:
   - Under the battery, plug the cable into the USB-C port.
3. **Replace the back cover**:
   - The device has a **groove on the left side** for the charging cable to fit through.
   - Snap the cover back into place with the cable running through the groove.

CHARGING THE DX8000 DEVICE

1. A **USB-C port** is located on the left side of the device for charging.
2. Connect the charging cable to this port until the device is fully charged.

4. Smart POS API Updates

Initialize

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

This section describes the parameters required for initialising the SDK for use with Smart POS, and the parameters that will cause errors if used.

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

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

Parameters Required for a Smart POS Initialize

The following parameters are required to initialize with Smart POS integration:

- `ParameterKeys.Password` — database encryption password.
- `ParameterKeys.IsSmartPosIntegration`
- `TRUE` enables Smart POS integration, allowing use of the Axium device only.
- `FALSE`, or omitting the parameter, enables the standard device integration, which allows the use of all other supported PIN pads.



`GetAvailablePinPads` returns only the PIN pads supported by the currently enabled integration. If `IsSmartPosIntegration` is enabled, then only Axium devices will be returned in the response.

Set Properties

There are no additional requirements around the `setProperties` method required for Smart POS transactions beyond the usual:

- `TerminalId`
- `TerminalKey`
- External device info (retrievable from `getAvailablePinPads`)
- `ParameterKeys.PinPadName`
- `ParameterKeys.PinPadConnectionType`
- `Environment`

The following existing parameter keys will be accepted, but will have no effect when using a Smart POS integration:

- `ApplicationIdentifier`

Connect And Configure

`connectAndConfigure` can change the settings on the terminal's Admin Screen. The following parameter keys are accepted for Smart POS only, and need Transaction Initiator UI v1.4.0 or later. A setting that is left out keeps its current value.

- `ParameterKeys.KioskModeEnabled` — `ParameterValues.TRUE` or `ParameterValues.FALSE`. Kiosk mode locks the terminal to the Transaction Initiator UI. It keeps that screen 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. Merchants can also turn kiosk mode on from the Admin screen; `getStatus` reports the current setting.
- `ParameterKeys.NotificationPanelAllowed` — `ParameterValues.TRUE` or `ParameterValues.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, of 6 to 12 digits. It must not be the default passcode, the same digit throughout or a run of consecutive digits. The current passcode is not needed. Pass it only when changing the passcode.

How the settings are applied:

- If `KioskModeEnabled` or `NotificationPanelAllowed` is not `TRUE` or `FALSE`, or `NewAdminPasscode` is empty, `connectAndConfigure` returns a `Result` of `FALSE` straight away. Nothing is connected and `connectAndConfigureFinished` is not fired. Correct the value and call again. A passcode that breaks the passcode rules is reported later, in the callback.
- Otherwise the settings are applied once the terminal is connected, before `connectAndConfigureFinished`. Wait for it before starting a transaction.
- If `connectAndConfigureFinished` reports one of the admin settings errors listed under New Error Codes, the terminal is connected and can take payments; only the setting the error is about was not applied (see the order below). For any other error, check `getStatus` to see whether the terminal is connected and which settings were applied, then call `connectAndConfigure` again.
- The kiosk settings are applied before the passcode. If they fail, the passcode is not changed; if the passcode fails, the kiosk settings stay changed. `getStatus` reports the kiosk and notification panel settings. A passcode change succeeded only if `connectAndConfigureFinished` reports a `Result` of `TRUE`.
- If an application update requested with `ParameterKeys.ApplyFirmwareUpdate` in the same call does not complete, the settings are not applied. Call `connectAndConfigure` again.

Get Status

When the Smart POS device is connected, `getStatus` also returns:

- `ParameterKeys.KioskModeEnabled` — the current Admin Screen setting.
- `ParameterKeys.NotificationPanelAllowed` — whether the notification panel is allowed right now. With kiosk mode off this can differ from the Admin Screen's switch, so do not pass it back to `connectAndConfigure` unchanged.
- `ParameterKeys.KioskSettingsApplied` — whether the terminal has applied the kiosk and notification panel settings. `FALSE` while the terminal is busy or has just started, or staff are in Android Settings from the Admin screen; during an update this can last many minutes. The terminal applies the settings as soon as it can, without another call.
- `ParameterKeys.AdminPasscodeIsDefault` — `TRUE` while the terminal's default passcode, `141309`, is still in force because no passcode has been set yet. Set a passcode with `ParameterKeys.NewAdminPasscode` on `connectAndConfigure`.

These keys are not returned when the terminal is not connected, or its Transaction Initiator UI is earlier than v1.4.0.



The default administrator passcode, `141309`, must be changed. Every terminal starts with it, and the Admin Screen can open Android's settings. While `AdminPasscodeIsDefault` is `TRUE`, set a passcode with `ParameterKeys.NewAdminPasscode` on `connectAndConfigure`.

Get Available Currencies

`getAvailableCurrencies` returns the merchant's boarded currencies, as for other payment devices. Pass one of them as `ParameterKeys.Currency` to `startTransaction`. A currency is required. If a call is refused:

- `TMSUpdateRequired` — the currencies have not been read from the terminal yet. Call `connectAndConfigure` again.
- `CurrencyNotFound`, from `connectAndConfigure` or `requestTmsUpdate` — the merchant has no boarded currencies that the SDK supports, and the device is disconnected. Contact support to correct the merchant's boarding, then call `connectAndConfigure` again. Until then, `getAvailableCurrencies` returns `PinPadNotConfigured`.
- `CurrencyNotSet` — `startTransaction` was called without a currency.
- `CurrencyNotSupportedForPaymentMethod` — the currency is not one of the merchant's boarded currencies. `getCardDetails` checks a supplied currency in the same way.
- `RegistrationRequired`, from `getCardDetails` — call `connectAndConfigure` again.

With a Transaction Initiator UI earlier than v1.4.0, only GBP and USD are returned.

Start Transaction

This section describes the parameters required for a Smart POS transaction, and the parameters that will cause errors if used.

Parameters Required for a Smart POS Transaction

The following parameters are required to start a Smart POS transaction:

- `ParameterKeys.Amount`
- `ParameterKeys.UserReference`
- `ParameterKeys.PaymentMethod`
- Must be set to `ParameterValues.Card`.
- `ParameterKeys.TransactionType`
- `ParameterKeys.Currency`
- Retrievable from `getAvailableCurrencies`. See Get Available Currencies.

Parameters That Will Cause Errors for a Smart POS Transaction

Smart POS currently only supports a limited number of features compared to traditional POS devices, therefore the following parameters will return an error if included when starting a Smart POS transaction:

- `ParameterKeys.DelayOnlineProcessing` with `ParameterValues.TRUE`
- `ParameterKeys.TippingType` with `ParameterValues.OnDeviceTipping` or `ParameterValues.BothTipping`
- `ParameterKeys.PurchaseOrderNumber`
- `ParameterKeys.TaxAmount`
- `ParameterKeys.TransactionPOI` with `ParameterValues.TapToMobile`
- `ParameterKeys.OnDeviceTippingPrompt`
- `ParameterKeys.DynamicTippingAmounts`
- `ParameterKeys.DynamicTippingPercentages`
- `ParameterKeys.DynamicTippingHeader`

New Parameters Available for a Smart POS Transaction

The following parameters are available when starting a Smart POS transaction. If they are not passed, the default values specified below are used.

PARAMETERKEYS.TRANSACTIONAUTHPREFERENCE

Determines how the transaction authorisation is initially processed for Smart POS transactions and can have the following values (default is Online):

- `ParameterValues.Online` — attempts online authorization, declines if online fails.
- `ParameterValues.DeferredForced` — immediately processes transaction offline, skipping online authorization.
- `ParameterValues.OnlinePreferred` — attempts online authorization, processes transaction offline if online fails.

PARAMETERKEYS.OFFLINETRANSACTIONUPLOADMODE

Determines how offline transactions are uploaded for processing for Smart POS transactions and can have the following values (default is Automatic):

- `ParameterValues.Automatic` — background worker automatically processes offline transactions when network returns.
- `ParameterValues.Manual` — requires a call to `processOfflineRequest` providing the `ParameterKeys.UserReference` for the transaction to attempt to be uploaded to the backend.

PARAMETERKEYS.MAGSTRIPEDEBIT

Determines if Debit is enabled for magstripe Smart POS transactions and can have the following values (default is True):

- `ParameterValues.TRUE`
- `ParameterValues.FALSE`

PARAMETERKEYS.MAGSTRIPESIGNATURE

Determines if signature is enabled for magstripe Smart POS transactions and can have the following values (default is True):

- `ParameterValues.TRUE`
- `ParameterValues.FALSE`

PARAMETERKEYS.PARTIALAPPROVAL

Determines if partial approval is enabled for Smart POS transactions and can have the following values (default is False):

- `ParameterValues.TRUE`
- `ParameterValues.FALSE`

PARAMETERKEYS.TIPAMOUNT

As on-device tipping is not supported when using a Smart POS device, a tip amount can be passed into the start transaction request. This parameter key will accept the value in the same format as `ParameterKeys.Amount`.

PARAMETERKEYS.CARDENTRYTIMEOUT

The number of 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 also accepted by `getCardDetails`. If it is not supplied, or is blank, the device's configured default is used. It is ignored by a Transaction Initiator UI earlier than v1.4.0.

When the prompt times out, the `CardEntryTimedOut` transaction update is emitted and the transaction, or the `getCardDetails` call, finishes with `TimeOut`.

PARAMETERKEYS.FEATURETOKENS

Requests one-way card tokens, as for other payment devices (see One way Card Tokens in the API guide). Needs Transaction Initiator UI v1.4.0 or later. A value that is not valid feature token XML returns `FeatureTokensInvalid` before the card is read.

The tokens are returned in `ParameterKeys.CardTokens`, in the `ITransactionFinishedListener` and from `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 `ParameterKeys.FundingCardTokens`, in the same format. For an offline or deferred transaction, the tokens are generated when it is processed online; retrieve them with `getTransactionInformation`.

Process Receipt

`processReceipt` prints a receipt on the device's integrated printer when `ParameterKeys.PrintReceipt` is `ParameterValues.TRUE`. Printing requires ARC 23.06.01 or later and Transaction Initiator UI v1.4.0 or later. With an earlier Transaction Initiator UI the print request is accepted but fails in the `IProcessReceiptFinishedListener` callback.

`ParameterKeys.ReceiptText`, `ParameterKeys.ReceiptSourceName` and `ParameterKeys.UserReference` are required. The printer prints 32 characters per line.

A `Result` of `TRUE` means the print request was accepted. The outcome is delivered to the `IProcessReceiptFinishedListener` as `ParameterKeys.ReceiptPrintResult`, with `ParameterKeys.Errors` on failure. If `ParameterKeys.PrintReceipt` is not `TRUE`, nothing is printed, `processReceipt` returns a `Result` of `TRUE` and no callback is fired.

After a print request, wait for the `IProcessReceiptFinishedListener` callback before starting a transaction; until then, other operations are refused with `AnotherOperationBeingProcessed`.

Email and SMS receipts are not supported on Smart POS.

Get Transaction Information

With Transaction Initiator UI v1.4.0 or later, if a sale was approved but its confirmation could not be completed straight away, the sale is still reported as `Committed`, and the device completes the confirmation later. Until it does, `getTransactionInformation` returns `ParameterKeys.OfflineStatus` set to `ParameterValues.OfflinePending`. Once the confirmation has been processed, it returns `ParameterKeys.OfflineCompletionState` set to `ParameterValues.OfflineCompletionProcessed`. If the sale is voided before then, its `TransactionState` is `Voided`.

Methods Requiring Smart POS Device Connection

The following API methods will require a connection to the Smart POS payment device in order to succeed, unlike when used in a standard device integration:

- **ConfirmTransaction**
- **VoidTransaction**
- **LinkedRefundTransaction**
- **GetTransactionInformation**
- **GetStatus** (for a full set of parameter keys to be returned)
- **ProcessReceipt** (when printing)

5. Events

Transaction Update Events

This section describes the events returned by the SDK during operation of Smart POS, including those for error codes and transaction events.

TransactionUpdate

The following new event is emitted during the transaction process:

`AttemptingToReconnect` indicates that the Payment Device SDK has lost connection to the Smart POS device and is attempting to reconnect to retrieve the transaction result.

`CardEntryTimedOut` is emitted when the card entry prompt times out (see `ParameterKeys.CardEntryTimeout`).

New Error Codes

This section details the new error codes added as part of the Smart POS integration. A full list of error codes can be viewed within the generated web documentation supplied within the release bundle's `doc` directory. Navigate to `index.html` then, under Modules, select `ChipDnaMobileErrorCodes`.

Connect And Configure

The following error codes may be returned for the admin settings straight away by `connectAndConfigure`. Nothing is connected and no callback is fired:

- `AdminSettingsNotSupported` — indicates that the integration is not Smart POS. Do not pass the admin settings to a PIN pad; the PIN pad is not connected.
- `KioskModeEnabledInvalid` — indicates that the value passed with `ParameterKeys.KioskModeEnabled` is not `TRUE` or `FALSE`.
- `NotificationPanelAllowedInvalid` — indicates that the value passed with `ParameterKeys.NotificationPanelAllowed` is not `TRUE` or `FALSE`.
- `ParameterMissing` — indicates that `ParameterKeys.NewAdminPasscode` was empty.

The following admin settings errors may be returned in the `connectAndConfigureFinished` callback. The terminal is connected; only the setting the error is about was not applied. Use `getStatus` to see the current kiosk and notification panel settings:

- `AdminSettingsNotSupported` — indicates that the terminal's Transaction Initiator UI is earlier than v1.4.0. Update it; retrying won't help.
- `AdminPasscodeNotDigits`, `AdminPasscodeTooShort`, `AdminPasscodeTooLong`, `AdminPasscodeDefaultNotAllowed` and `AdminPasscodeTooSimple` — indicate that the value passed with `ParameterKeys.NewAdminPasscode` does not meet the passcode rules.
- `AdminPasscodeRefused` — indicates that the terminal refused the new passcode.

Other errors in the callback, such as `ConnectionFailed`, `DeviceBusy` or `AccessDenied`, mean the settings may not have been applied. Check `getStatus` to see whether the terminal is connected and which settings were applied, then call `connectAndConfigure` again.

Process Receipt

The following error codes may be returned straight away by `processReceipt`, with a `Result` of `FALSE`. No callback is fired for them:

- `ReceiptSourceEmailNotSupported` and `ReceiptSourceSMSNotSupported` — indicate that an email or SMS receipt was requested. Only printing is supported on Smart POS.
- `SourceNameMissing`, `ReceiptTextMissing` and `UserReferenceMissing` — indicate that a required parameter was not supplied.
- `ConnectAndConfigureRequired`, `PinPadNotConnected` or `PinPadNotConfigured` — indicate that the Smart POS device has disconnected (call `connectAndConfigure`), is not connected, or is not configured.
- `AnotherOperationBeingProcessed` — indicates that another operation is in progress.

The following error codes may be returned in the `IProcessReceiptFinishedListener`, for example:

- `PrinterPaperEmpty` — indicates that the printer is out of paper.
- `PrinterError` — indicates that the printer reported an error. Check the printer before retrying.
- `DevicePrinterNotSupported` — indicates that the device has no integrated printer, or that its ARC version does not support printing.
- `DeviceBusy` — indicates that the printer is busy.
- `PinPadNotConnected` — indicates that the terminal is not connected.
- Other failures are reported with the usual server and connection errors, such as `AccessDenied`, `ServerResponseError`, `TimeOutError` or `ConnectionFailed`.

Start Transaction

The following new error codes may be returned during calls to `startTransaction`:

- `TransactionAuthPreferenceInvalid` — indicates that the value passed with `ParameterKeys.TransactionAuthPreference` is invalid.
- `OfflineTransactionUploadModeInvalid` — indicates that the value passed with `ParameterKeys.OfflineTransactionUploadMode` is invalid.
- `CardEntryTimeoutInvalid` — indicates that the value passed with `ParameterKeys.CardEntryTimeout` is not a whole number from 1 to 3600. Also returned by `getCardDetails`.

Transaction Finished

The following new error codes may be returned in the `transactionFinishedListener`:

- `TransactionInformationUnavailable` — indicates the transaction information could not be retrieved due to a connection failure with the Smart POS device.



The `AUTO_CONFIRM` parameter is not supported for Standalone Refunds on Axium devices. You should refer to `TRANSACTION_RESULT` and `TRANSACTION_STATUS` values returned in the Transaction Finished and Transaction Information events to determine whether a `ConfirmTransaction` call is expected to succeed. Transactions with a `TRANSACTION_STATUS` of `Committed` are already confirmed and will not succeed if `ConfirmTransaction` is called. `ConfirmTransaction` calls on a transaction with a `TRANSACTION_RESULT` of `Declined` will also fail.

6. Offline Transaction Queue Processing

Only transactions associated with the currently configured environment can be processed in the offline queue. Therefore if configured on Staging, only offline transactions performed against Staging credentials will be able to be processed. Likewise if configured on Live, only offline transactions performed against Live credentials will be able to be processed.

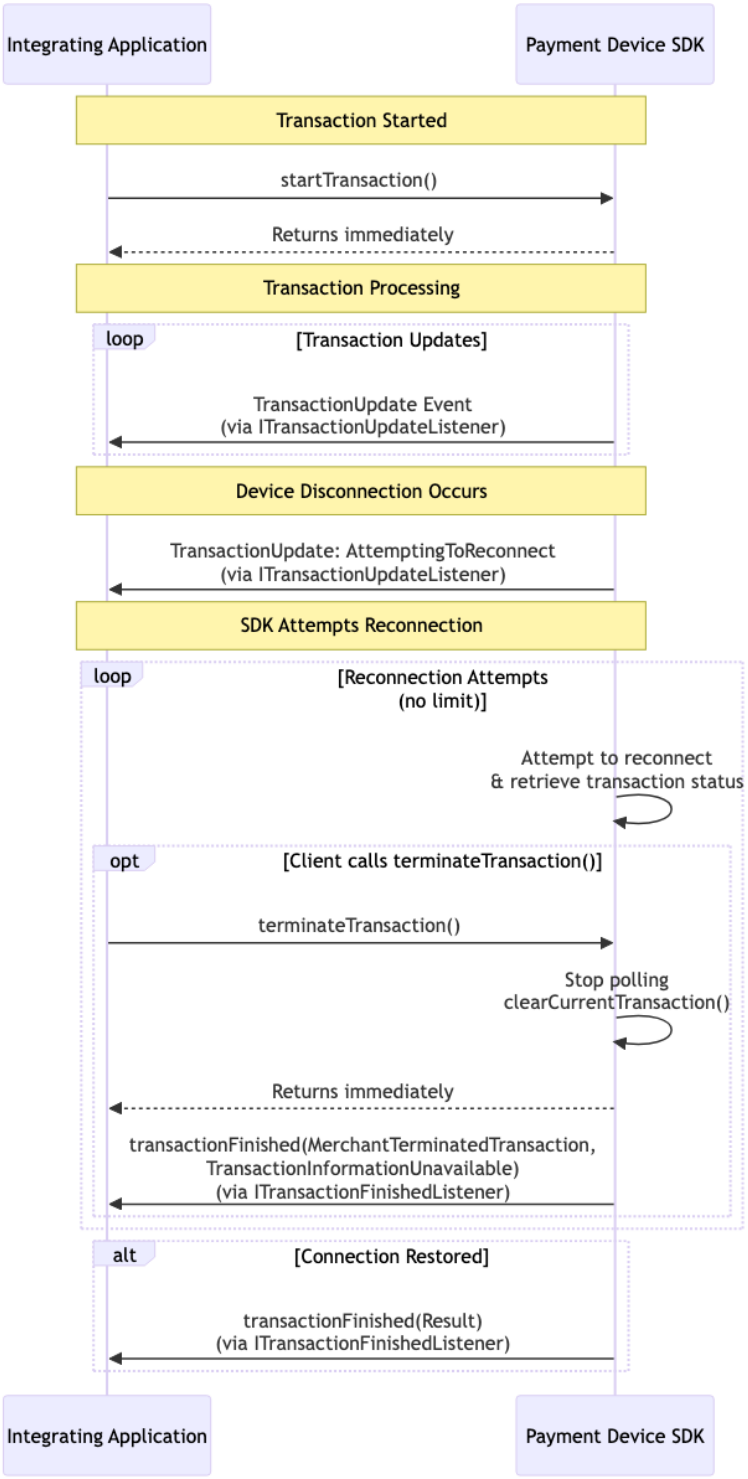
The environment associated with a transaction can be found by calling `getTransactionInformation` for the specified transaction and looking at the value associated with `ParameterKeys.Environment`. The currently configured environment can be found by calling `getStatus`, again looking at the value associated with `ParameterKeys.Environment`.

7. Device Disconnection During a Transaction

As described in the high level overview, transaction processing is fully performed on the Ingenico Axium device. Therefore if a disconnection between the Payment Device SDK and device occurs during a transaction, a termination does not automatically occur, as it would have done with a non Smart POS integration.

If a disconnection between the Payment Device SDK and device occurs during a transaction, then the Payment Device SDK will attempt to reconnect to the Axium device, until a reconnection is made or the `terminateTransaction` API method is called by the integrating application, as shown in the sequence diagram below.

Diagram 3 – Sequence diagram of a device disconnection during a transaction


Integrating Application
Payment Device SDK
Transaction Started
startTransaction()
Returns immediately
Transaction Processing
loop
[Transaction Updates]
TransactionUpdate Event
(via ITransactionUpdateListener)
Device Disconnection Occurs
TransactionUpdate: AttemptingToReconnect
(via ITransactionUpdateListener)
SDK Attempts Reconnection
loop
[Reconnection Attempts
(no limit)]
Attempt to reconnect
& retrieve transaction status
opt
[Client calls terminateTransaction()]
terminateTransaction()
Stop polling
clearCurrentTransaction()
Returns immediately
transactionFinished(MerchantTerminatedTransaction,
TransactionInformationUnavailable)
(via ITransactionFinishedListener)
alt
[Connection Restored]
transactionFinished(Result)
(via ITransactionFinishedListener)
Integrating Application
Payment Device SDK

If the `terminateTransaction` API method is called by the integrating application during the disconnection, the `transactionFinished` event will be returned including the following parameters:

- `TransactionResult = Unknown`
- `TransactionState = Unknown`
- `Errors = MerchantTerminatedTransaction, TransactionInformationUnavailable`

When this occurs, the integrating application will need to call `getTransactionInformation` once connection to the Ingenico Axium device has been restored to see the final outcome of the transaction.

This applies when the integrating application connects to the Axium over Bluetooth. When it runs on the Axium itself (TCP/IP), `terminateTransaction` asks the Transaction Initiator UI to cancel the transaction and waits for the outcome, so it does not end a transaction while the Transaction Initiator UI is not answering. See the Axium Development Guide Supplement.

8. Retrieving Log Files

When integrated into a Smart POS device, local log files can be retrieved to aid integration support in diagnosing issues.

The below code snippet gives the file location of logs for Smart POS integrations.

```
String chipDNASmartPosLogPathComponent =
        "NMI" + File.separator + "ChipDNA" + File.separator + "smartpos_logs" +
File.separator;
```

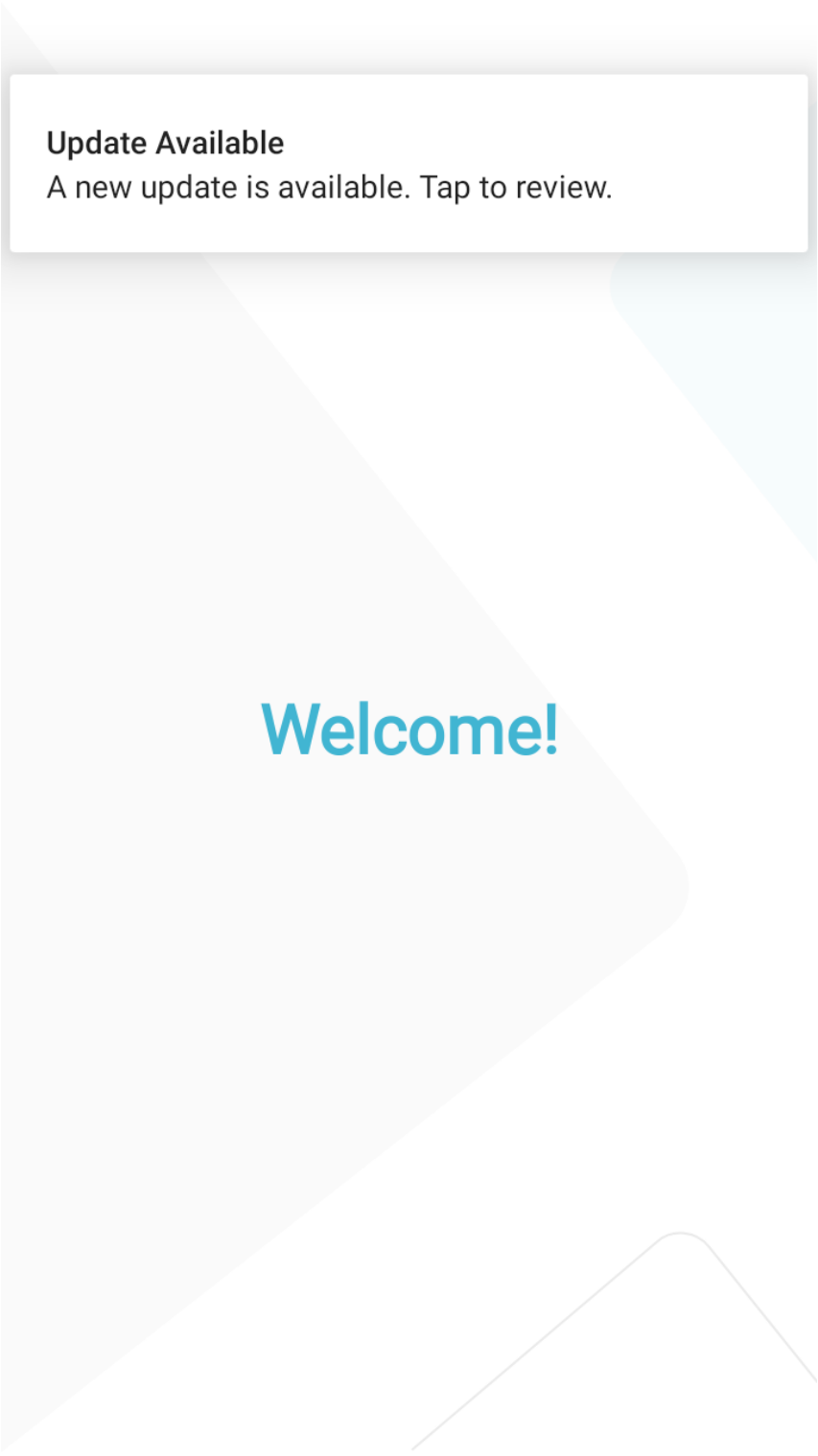
9. OTA Application Updates

NMI has implemented over-the-air (OTA) application updates. As part of this, a new NMI Device Agent application will be installed on your device. This has been designed to align with the existing Miura OTA update experience while leveraging the additional capabilities of Smart POS devices.

When an OTA update becomes available, your application will receive a Firmware Update event. A notification will also be displayed on the device. You can either trigger the update

programmatically via the SDK or initiate it directly on the Axium device by tapping the notification.

Triggering Update via Axium Device

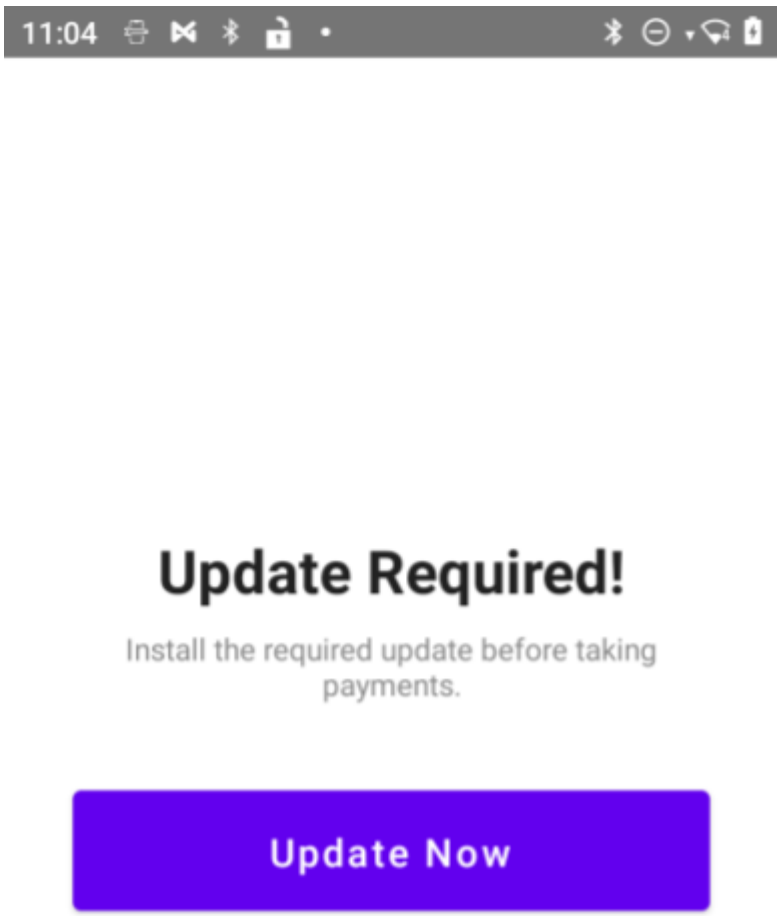


As with the ChipDNA Mobile SDK update process an update can either be of the type:

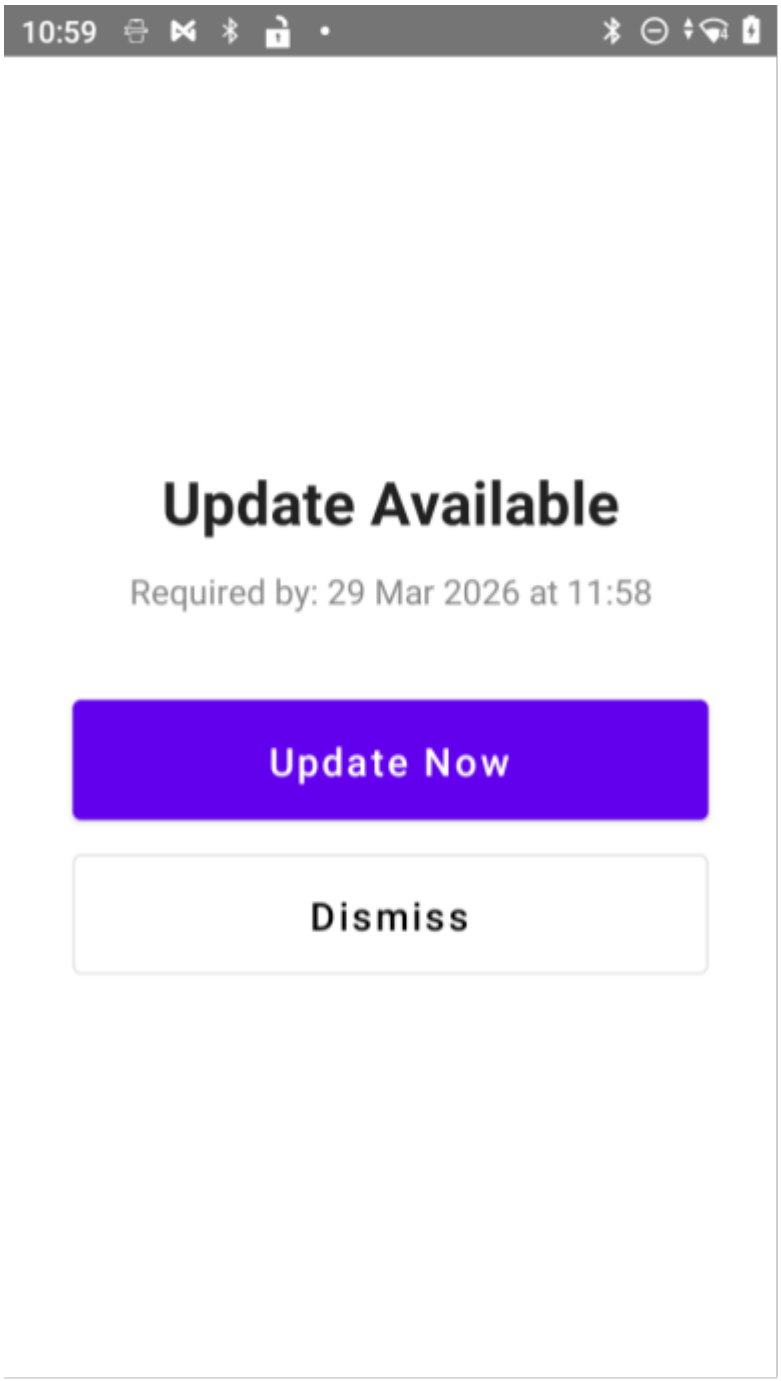
- **Required** — the update must be performed before transaction processing can continue.
- **Deferrable** — the update can be deferred for later but will expire eventually to become required.

There are two ways to trigger the update via the Axium device UI. You can either tap the notification, which will open the Device Agent application, or you can manually enter the Device Agent application and trigger the update from there.

Triggering Update via ChipDNA Mobile



Device Agent - Update Required



Device Agent - Update Available

When triggering the update using the ChipDNA Mobile SDK the NMI Device Agent application will take over and will immediately begin updating. Once the update has completed the previous app will automatically come to the foreground.

The ChipDNA Mobile API remains unchanged from existing Miura integrations. For implementation details, refer to the API Developer Integration Guide under:

- *Configuration and Utility Methods*, specifically *ConnectAndConfigure*
- *Configuration and Utility Events*, specifically *FirmwareUpdate Callback*

Large updates can take upwards of 5 to 10 minutes to complete. Unfortunately the Download and Install process are controlled by the Axium firmware, which doesn't return progress updates. If an update fails a screen will be displayed allowing you to retry the update.

Appendix 1 - Supported API Methods

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

Table 2 - 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	-
continueVoiceReferral	No	-

API Method	Supported	Notes
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	-
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 2 - Smart POS Unsupported Features

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

Table 3 - Supported Smart POS Features

Feature	Supported
Delay Online Processing	No
On Device Tipping	No
One Way Card Tokens	Yes

Appendix 3 - Axium Data Usage

This table details the network data consumed by an Axium terminal, broken down per merchant action, for use in sizing cellular data plans. Figures are on-the-wire bytes including TCP/IP and TLS overhead.

Table 4 - Axium data usage

Event	Data per occurrence
Sale	34.7 KB

Event	Data per occurrence
Auth	34.8 KB
Confirm	6.0 KB
Void	5.1 KB
Linked Refund	5.6 KB
Connect and Configure	187.8 KB