worldpay

triPOS© Mobile Android SDK Integration Guide

v4.4.0

August 19, 2025

Table of Contents

Document History

Revision Date	Comments
08.19.2025	Updated Document for triPOS® Android SDK 4.4.0 • Added Fee Assist (Hosted Credit Surcharge) support including new configuration flag 'isFeeAssistEnabled • Updated the developer engine URL under Network section • Updated the project dependencies to support Android 15 compatibility • Added preReadDataLifetimeSeconds field to Transaction Configuration. • Added sleepTimeoutSeconds to Device configuration • Added QR Code and Barcode Scanning feature detail • Added barcodeReaderEnabled to Device configuration • Added onPromptUserForCardWithBarcode in DeviceInteractionListener • Added QR Code and Barcode Scanning for valuetec Gift Cards • Added QR Code and Barcode Scanning Functionality in Sample App
05.08.2025	Updated Document for triPOS® Android SDK 4.3.0 • Added heartbeatEnabled, wifiRoamingEnabled and devicePool to Device configuration • Added swipe Attempts, contact Attempts, and contactless attempts fields to "Transaction Configuration". • Updated details on firmware update process for Ingenico Tetra devices • Updated details on card input retries. • Added Global, ChasePaymentech, FirstData and TSYS processors for Ingenico Moby devices. • Added Worldpay ROI payment processor for IngenicoAxium devices. • Added support for connection from an Android device to multiple Ingenico Moby5500 and Moby5500M payment devices
12.19.2024	Updated Document for triPOS® Android SDK 4.2.0
09.10.2024	Updated Dependencies and Clarity on ARC Setup

08.16.2024	Updated Document for triPOS® Android SDK 4.0.0
06.07.2024	Updated Document for triPOS® Android SDK 3.1.0
01.24.2024	Rebranding changes (images & logo)
12.11.2023	Updated Document for triPOS® Android SDK 3.0.0
07.26.2023	Updated Configuration for triPOS® Android SDK 2.2.0
03.14.2023	Added Ingenico UPP USB support related information
02.17.2023	Added Moby device support related information
02.09.2023	Initial Document Release for triPOS® Android SDK 2.0.0

triPOS® Mobile Android SDK

Overview

The Worldpay triPOS® Mobile SDK is designed to allow Android applications rapid integration for EMV support, point-of-interaction device support, and Worldpay Integrated Payments platforms.

In the North American (NA) region this is the *Express* platform whilst in the UK this is the *Access 4 POS* (*A4P*) platform.

While the SDK has been certified for EMV and to these platforms, the application to which it is being integrated must also complete testing with Worldpay Integrated Payments and the chosen platform to ensure all necessary fields are populated with the correct data and receipts are generated with the required information. Android v10 and above are supported the Worldpay triPOS® Mobile SDK.

Transaction Flow and Component Interaction

triPOS® accepts requests from business management software for processing end-to-end financial transactions through the Worldpay gateway. The diagram below illustrates the interaction among the business management software, triPOS®, the PIN pad and Worldpay platform.

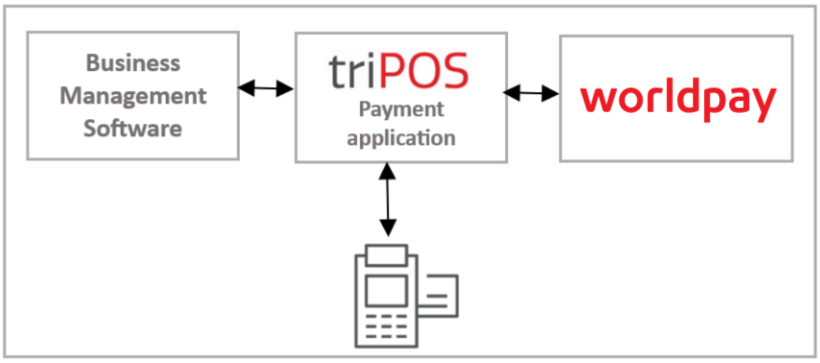
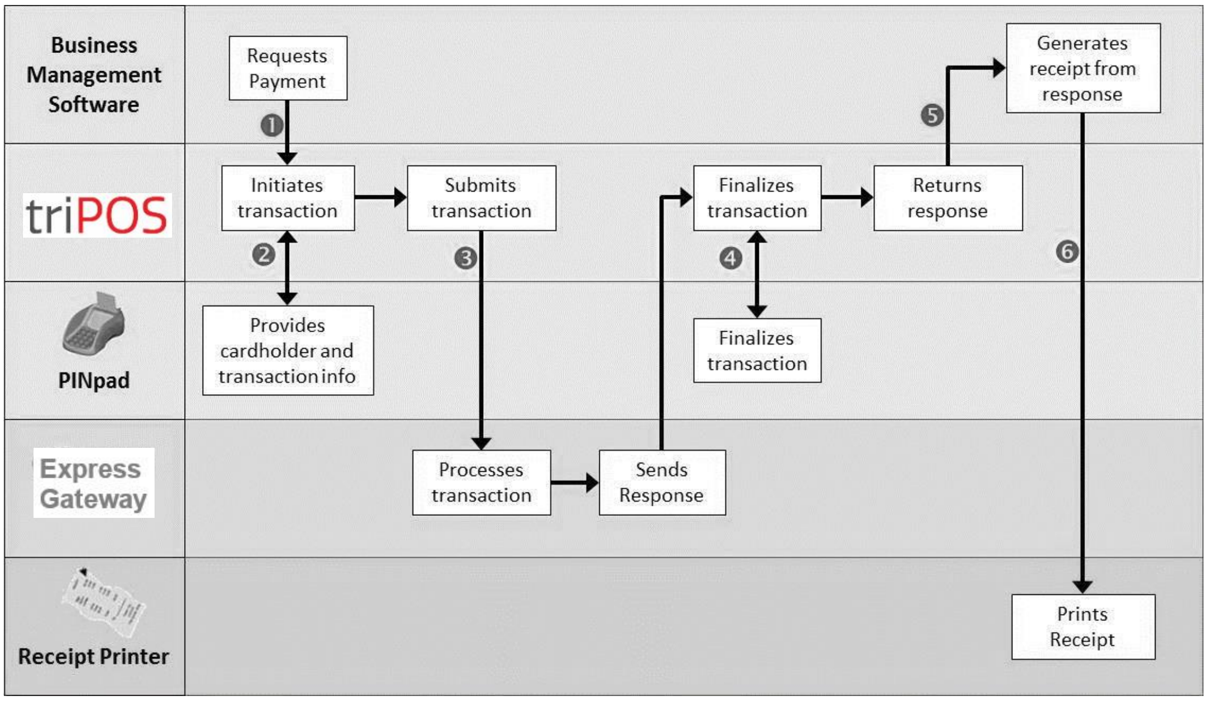


Figure 1: Component Interaction

The graphic below illustrates a typical transaction flow through the system in the NA region:



1. The **business management software** sends a payment request to triPOS®.
2. triPOS® interfaces with the **PIN pad** to obtain the card information, manages cardholder selection of options and verification, then collects the EMV tags required to complete the transaction.
3. If applicable, triPOS® sends the transaction to the ***Express*** platform to be forwarded to the card issuing bank for an authorization attempt.
4. Upon receiving a response from the *Express* platform, triPOS® proceeds with final validation of the transaction through the PIN pad.
5. triPOS® returns a response in the same format as the request containing data that the business management software can use to create receipts for the merchant and cardholder.
6. The business management software prints the receipts utilizing the data returned by triPOS®.

Network

The Android device being used requires a data connection. The data connection does not have to be always available, but it does have to be available at the time a transaction is processed. This connection may be Wi-Fi, mobile data, or any other data connection available that allows an HTTPS connection to the host.

You must have credentials assigned by Worldpay to use this SDK. For development and certification, you may sign up for a free test account by visiting https://docs.worldpay.com.For production use, contact your Worldpay account representative.

Certifications

EMV Certifications

For each payment device supported in the application, Worldpay triPOS® Mobile Android SDK Framework must be certified for use with the selected payment processor. Please reference the triPOS® Mobile Android SDK release notes for the version you are using.

Getting Started

Add reference to triPOSMobileSDK-release.aar to the Android project and refer to the instructions below to add necessary firmware files and steps to use the SDK.

Using the SDK

The Worldpay triPOS® Mobile SDK Framework is used by importing it and then accessing the SDK functionality through the "sharedVtp" singleton. The SDK must be initialized with the required configuration before performing any functions. Before utilizing any function that interacts with the device, the application must wait for the device to become connected.

☞ Note: if necessary, the triPOS® Mobile SDK will update device firmware and settings. If a firmware update is required, the first initialization may take between 10 - 45 minutes or more depending on the device family and the connection type.

Device Setup

Ingenico Tetra

Firmware Updates

Each Worldpay triPOS® Mobile SDK Framework version requires Ingenico payment device firmware updates. For each supported Ingenico Tetra device, add the firmware file(s) (*.OGZ for

UPP devices) in the application's assets. Firmware files are in the **Firmware Files** folder, in subfolders named after each supported device.

The process to update Ingenico Tetra firmware depends on your device's current firmware version. Following Ingenico guidelines, updates require clearing old JCB kernels and downloading OGZ files.

For low-storage devices including Lane 3000, Lane3600, Link 2500, and Move 5000:

- To support firmware updates, download a backport file first. Then, download the firmware file parts in the order listed in the tables in each applicable section below. The device reboots once after the backport file is downloaded and again after all firmware file parts are installed.

Update UPP Firmware (US Only)

To upgrade older UPP versions, follow the firmware upgrade procedures in the order outlined below, starting from the applicable firmware version on your device. Continue this process for each subsequent version until you reach the latest version.

☞ Note: The SDK automatically rewrites firmware components if a firmware update is interrupted.

Each step will require a device reboot upon completing installation or file write.

This section describes how to update an Ingenico Tetra device to v6.81.11.

The table lists the applicable Ingenico Tetra devices and their respective files for upgrade.

Link 2500	Move 5000	Lane 3000	Lane 3600	All Devices
UTKI78P0783491.OGZ	UTKI71P0783491.OGZ	UTKI83P0783491.OGZ	UTKI91P0783491.OGZ	U*xx*P068120.OGZ
UTKI78P0783492.OGZ	UTKI71P0783492.OGZ	UTKI83P0783492.OGZ	UTKI91P0783492.OGZ	UPP6_81_xx_JCBCleaner.OGZ
UTKI78P0783493.OGZ	UTKI71P0783493.OGZ	UTKI83P0783493.OGZ	UTKI91P0783493.OGZ	Backport68155.OGZ
UTKI78P0783494.OGZ	UTKI71P0783494.OGZ	UTKI83P0783494.OGZ	UTKI91P0783494.OGZ	

☞ Important! After each step in the following procedure, you must wait for the system to reboot.

Update Ingenico Tetra Device from v6.81.11 to v7.83.49

This section describes how to update an Ingenico Tetra device to v7.83.49.

The table lists the applicable Ingenico Tetra devices and their respective v7.83.49 files.

Link 2500	Move 5000	Lane 3000	Lane 3600	All Devices
UTKI78P0783491.OGZ	UTKI71P0783491.OGZ	UTKI83P0783491.OGZ	UTKI91P0783491.OGZ	UPP6_81_xx_JCBCleaner.OGZ
UTKI78P0783492.OGZ	UTKI71P0783492.OGZ	UTKI83P0783492.OGZ	UTKI91P0783492.OGZ	Backport68155.OGZ
UTKI78P0783493.OGZ	UTKI71P0783493.OGZ	UTKI83P0783493.OGZ	UTKI91P0783493.OGZ	_Patch.TGZ
UTKI78P0783494.OGZ	UTKI71P0783494.OGZ	UTKI83P0783494.OGZ	UTKI91P0783494.OGZ	

☞ Important! After each step in the following procedure, you must wait for the system to reboot.

To update a device from v6.81.xx to v7.83.49:

1. Download the UPP6_81_xx_JCBCleaner.OGZ file.
2. Download the Backport68155.OGZ file.
3. Download each firmware file below in the following order:
    a. UTKIxxP0783491.OGZ
    b. UTKIxxP0783492.OGZ
    c. UTKIxxP0783493.OGZ
4. Perform a file write of _Patch.TGZ.
   Wait for the system to reboot to complete the update process.

Update Ingenico Tetra Device from v7.8x.00 to 7.83.49 or higher

This section describes how to update an Ingenico Tetra device to v7.83.49 or higher.

The table lists the applicable Ingenico Tetra devices and their respective files.

Link 2500	Move 5000	Lane 3000	Lane 3600	All Devices
UTKI78P0783491.OGZ UTKI78P0783492.OGZ UTKI78P0783493.OGZ UTKI78P0783494.OGZ	UTKI78P0783491.OGZ UTKI78P0783492.OGZ UTKI78P0783493.OGZ UTKI78P0783494.OGZ	UTKI83P0783491.OGZ UTKI83P0783492.OGZ UTKI83P0783493.OGZ UTKI83P0783494.OGZ	UTKI91P0783491.OGZ UTKI91P0783492.OGZ UTKI91P0783493.OGZ UTKI91P0783494.OGZ	UPP6_81_xx_JCBCleaner.OGZ Backport79945.OGZ _Patch.TGZ

☞ Important! After each step in the following procedure, you must wait for the system to reboot.

To update a device to v7.8x.31 or higher:

- For all Ingenico Tetra devices:
  1. Download the Backport79950.OGZ file.
  2. Download each firmware file below in the following order:
     i. UTKIxxP0783491.OGZ
     ii. UTKIxxP0783492.OGZ
     iii. UTKIxxP0783493.OGZ
     iv. UTKIxxP0783494.OGZ
  3. Perform a file write of _Patch.TGZ.
     Wait for the system to reboot to complete the update process.

Note: The backport files update the terminals to an intermediate firmware version (6.81.55 or 7.99.45). The devices should not be used for any payment transactions while on this version and should be updated to 7.83.49.

Menu Access on the IngenicoUpp Devices

Accessing the Ingenico Tetra menu access while the IngenicoUpp device is offline, in idle, or connected to the SDK:

- 0 0 0 0 - System information
- 0 0 0 1 - Communication admin menu
- 2 6 3 4 - Tetra system menu

Ingenico UPP USB Pairing

When connecting to an Ingenico UPP device using USB, the terminal's communication type must be updated to USB_CDC from terminal's admin menu. Admin menu can access by 0 0 0 1.

The USB Access Permission dialog will prompt every time when the device connects, disconnects, reconnects or reboots. To grant permission for a device until the app is uninstalled or data is deleted through the app manager, enable the check box to 'Always remember the device' within the USB Access Permission dialog. The Android System will take care of the permission for connections, reconnections or reboots.

Ingenico Axium

To properly enable an Ingenico Axium device, ensure the required Ingenico Axium Retail Core (ARC) libraries are added to the project (See dependencies). When connecting to an Ingenico Axium device, the first step is setting the DeviceConfiguration.deviceType to "IngenicoAxium". Upon initialization, triPOS will download the required configuration and EMV configuration files onto the Axium device. To apply the changes from these files, the ARC service will automatically reboot and re-connect to the SDK. Once the DeviceConnectionListener returns the onConnected callback, the device is ready to be used.

Ingenico Moby 5500

Firmware Files

Each version of the Worldpay triPOS® Mobile SDK Framework, if used with any Ingenico payment devices, requires that the payment device be updated to the correct firmware. For each Ingenico device that is supported, the firmware file (with extension *.uns for RUA Moby devices) must be added directly to the assets directory in the target application. The correct firmware file(s) to add can be found in a subfolder with the same name as the supported device in the "Firmware Files" folder. The firmware files for Ingenico Moby5500/5500M are different for CL1 (firmware v15.01) and CL3(firmware v16.02). The appropriate the firmware files need to be included based on the device models supported.

Bluetooth Pairing

When connecting to a Moby device for the first time, the triPOS® SDK attempts to pair and connect to the device. Upon finding the device to pair, the SDK initiates pairing and android prompts to confirm device pairing. After confirming, a temporary pairing is established with the device. At this point the Bluetooth button (on the side of the device) needs to be pressed to proceed. The Moby device displays a random pattern of LED lights and a request to confirm is sent to the integrating application through the "OnConfirmPairing" method of the device connection listener that also includes the callback listener to confirm\decline connection. The pairing and connection are established once the confirmation is sent back to the SDK through the callback ("ConfirmCallbackListener") listener.

SleepTimeoutSeconds

The sleepTimeoutSeconds parameter allows integrator to control the sleep behaviour of the Moby5500 due to inactivity. Maintaining a stable connection with the triPOS Mobile SDK, as the device disconnects when it sleeps and requires reinitialization to reconnect.

By default, the Moby5500 enters sleep mode after 180 seconds (3 minutes) of inactivity. To customize this behavior, integrators can set the sleepTimeoutSeconds parameter to any value between 0 and 3600 seconds:

Setting it to 0 seconds disables sleep mode entirely, keeping the device awake indefinitely. This is not recommended by Ingenico, as it can significantly reduce battery life.

Any value between 1 and 3600 seconds will define the duration of inactivity after which the device sleeps.

If a value greater than 3600 is provided, the SDK will automatically cap it at 3600 seconds.

Note

- If Integrator does not want to reconnect after sleep time the heartbeatEnabled flag should be disabled from device configuration.

Ingenico Telium

Firmware Files

Each version of the Worldpay triPOS® Mobile SDK Framework, if used with any Ingenico payment devices, requires that the payment device be updated to the correct firmware. For each Ingenico Telium device that is supported, the firmware file (with extension *.OGZ for RBA devices) must be added directly to the assets directory in the target application. The correct firmware file(s) to add can be found in a subfolder with the same name as the supported device in the "Firmware Files" folder.

BBPOS Chipper 3X BT

Bluetooth Pairing

When a user first attempts to connect to a BBPos Chipper 3X BT device via the triPOS® Mobile SDK Framework, they will be prompted to enter a 6-digit code. This code is located on the back of the BBPos Chipper 3X BT device and is labelled BT Passkey. After the code is correctly entered, the BBPos Chipper 3X BT device will be paired to the Android device.'

triPOS® Mobile SDK Initialization and Connectivity

Overview

The triPOS® Mobile SDK is designed to be initialized once per application session and only deinitialized when the application shuts down or if a configuration change requires it (e.g. changing the device type or its connectivity type). After the SDK is initialized, it continues to attempt to connect to the device using the configured device type, connectivity option, and other communication parameters. When the device is connected and initialized, the SDK notifies the application by calling the device connection protocol method after which, if the device disconnects, the SDK notifies the application by calling the device disconnection protocol method. During initialization, the SDK may also notify the application of errors using the device error protocol method.

Device Connection and Initialization

When the triPOS® Mobile SDK is initialized, it begins attempting to connect to the selected device (through Bluetooth identifier or through IP address and port number or through USB) and continues to do so until a device connects, or the SDK is deinitialized. There are two parts to the device connection, one in which the SDK recognizes the device is connected and one in which the SDK notifies the DeviceConnectionListener.

SDK Connection

When the triPOS® Mobile SDK recognizes a device has connected, the SDK first attempts to identify the type of device to ensure it is a supported model. The SDK next checks to make sure the device is setup correctly to work with the triPOS® Mobile SDK. This check includes firmware versions, P2PE enablement, injected keys, forms, and configuration. Except for injected keys, if the SDK identifies any of these are missing or incorrect, it attempts to load them in the following order:

1. Firmware – The software that runs on the device and depending on the device model, connection type, and connection quality, may take 20+ minutes to update.

2. Forms – Forms and images used by the triPOS® Mobile SDK for transaction processing.

3. Custom prompts – Secure and signed prompts for clear text input that are not part of the default firmware package.

4. P2PE enablement – Ensures P2PE is enabled and configured for the correct type and options.

5. EMV configuration – Settings for both contact and contactless EMV transaction processing.

6. Date/time – Set the correct local date and time on the device.

7. Other configuration – General settings for transaction processing and general operation.

> ☞ Note: During this process, the device may reboot for any or all the above steps. If the device reboots, the SDK automatically reconnects to the device and continues with the next step.

Initialize

The initialize method is intended to be called one time per application session. If the initialization is successful, the triPOS® Mobile SDK attempts to start the connection with the configured device and continues to do so until the device is connected, or the SDK is deinitialized.

- VTP.initialize(Context applicationContext, Configuration configuration, DeviceConnectionListener deviceConnectionListener)
    - Initializes triPOS® Mobile SDK using the passed in application context, configuration, and device connection listener.
- VTP.initialize(Context applicationContext, Configuration configuration, DeviceConnectionListener deviceConnectionListener, OTAUpdateListener otaUpdateListener)
    - Initializes triPOS® Mobile SDK using the passed in application context, configuration, and device connection listener and over-the-air device update listener.

Initialization Progress – Ingenico Tetra\Ingenico Axium

On supported devices, triPOS Mobile SDK passes back the progress of the device initialization through the initialization progress listener if one is provided. For Ingenico Tetra and Ingenico Axium devices, this includes various progress updates including file downloads to device (firmware, configuration etc) as well as other updates like device reboots, and configuration updates.

- VTP. setInitializationProgressStatusListener(InitializationProgressStatusListener listener)

UpdatingFirmware	The device's firmware is being updated. When applicable, this status is sent multiple times indicating the progress of the file download.
UpdatingFormFiles	The forms package on the device is being updated. When applicable, this status is sent multiple times indicating the progress of the file download.

UpdatingCustomPrompts	The prompts package on the device is being updated. When applicable, this status is sent multiple times indicating the progress of the file download.
UpdatingIdlePromptFiles	The idle image on the device is being updated. When applicable, this status is sent multiple times indicating the progress of the file download.
ConfiguringP2pe	The P2PE package on the device is being updated. When applicable, this status is sent multiple times indicating the progress of the file download.
ConfiguringEmv	The device's EMV settings are being configured.
SettingClock	The device's clock is being set.
RebootingDevice	The device is being rebooted. This status may be sent multiple times during the initialization process.
UpdatingFiles	Indicates a file is being downloaded to the device. This may include any file download that is not listed earlier like whitelist file, or other configuration file.
UpdatingWifiRoamingFile	The WIFI roaming configuration on the device is being updated.

Initialization Progress – Ingenico Moby

For Ingenico Moby and BBPos devices, triPOS Mobile SDK will use the OTA update listener to pass back initialization progress including firmware and configuration updates. This is an optional listener provided while initializing the SDK.

Firmware file	The firmware update progress is sent back through the OTA update listener using the "onOverTheAirUpdateProgress" callback method.

	Type - OTAUpdateType.Firmware CurrentProgress – Firmware update progress percentage
Whitelist file	The firmware update progress is sent back through the OTA update listener using the "onOverTheAirUpdateProgress" callback method. Type - OTAUpdateType.Firmware CurrentProgress – Firmware update progress percentage
Configuration File	The EMV configuration files are pushed to the device during initialization. The file download progress is sent back through the OTA update listener using the "onOverTheAirUpdateProgress" callback method. Type : OTAUpdateType.Config CurrentProgress: Configuration file update progress percentage.

Deinitialize

Deinitializes triPOS® Mobile SDK and disconnects the POI device. The deinitialize method is intended to stop any device activity ongoing and shut down the SDK.

- VTP.deinitialize()

DeviceConnectionListener Callback Protocol

When the triPOS® Mobile SDK has connected and configured a device, the SDK notifies any VTP listeners the device is connected and ready for use using the onConnected protocol method. If the device requires updates or configuration, the SDK notifies any VTP delegates using the device initialization progress protocol method. If any part of the SDK connecting and configuring the device fails, the SDK notifies any VTP delegates about the error using the device error protocol method.

Troubleshooting connection issues

In the event the triPOS® Mobile SDK cannot connect to the device:

1. Ensure the proper device type, connection type, and communication parameters are correct in the SDK.

2. Ensure the device is powered on.
3. For battery powered devices, ensure the battery is sufficiently charged.
4. For Ethernet devices, ensure the device is connected to an active network and is configured properly for that network.
5. For Wi-Fi devices, ensure the device is connected to an active access point, has good signal strength, and is configured properly for that network.
6. For Bluetooth devices, ensure the device is in proximity to the POS terminal.
7. For Bluetooth devices that require pairing, ensure the device is paired with the POS terminal using the POS terminal's Bluetooth settings/utilities.
8. For USB devices, ensure the device is connected to the POS terminal using USB cable, and is allowed USB access permission for connecting the POS terminal.
9. Power off the device and power it back on.

Retrieving triPOS® Mobile SDK logs

The triPOS® Mobile SDK leverages log4j for both console and file logging.

- Log files will be automatically stored in the internal memory of the device at path: Android/data/integrator_app_packagename/files/fis/tripossdk/log/*.log.

Normal Operation

While the triPOS® Mobile SDK attempts to always keep the device connected, it may not always immediately detect a device has disconnected or in some cases, not at all. Prior to beginning a flow (e.g. sale) or an action (e.g. card input), the application may use Device.isConnected to determine if the device is connected or check any returned errors from the operation for an indication the device is not connected. In the event the SDK cannot automatically reconnect to a device, follow the steps in the No Connection or Connection Errors section above. In some cases, it may be necessary to deinitialize and reinitialize the SDK to reestablish the connection.

Device Heartbeat

The triPOS Mobile SDK includes a robust heartbeat mechanism to maintain device connectivity for Ingenico Tetra and Ingenico Moby devices. Once the SDK is initialized, it starts sending periodic heartbeats to monitor the connection status. If the heartbeat detects a disconnection, the SDK attempts to reconnect using the original connection parameters, retrying approximately every 15 seconds. If the connection cannot be re-established within 10 consecutive attempts, the

SDK triggers the "onError" method of the device connection listener, sends an exception, and automatically deinitializes itself, ceasing further heartbeats.

Heartbeat could be enabled\disabled through the "heartbeatEnabled" flag under device configuration. When heartbeats are disabled, triPOS Mobile SDK will not attempt to reconnect to the device in case of any connection failures.

triPOS® Configuration

Application Configuration

applicationMode	The enumeration for application mode. This value determines which endpoint is used for *Express* processing. • Production - used for production • TestCertification - used for testing/certification
binWhitelistFile	An optional BIN whitelist file that needs to be loaded into the payment device. Applicable for Ingenico Telium, Ingenico Tetra and Ingenico Moby devices only. The whitelist file needs to be placed in the main application's asset directory.
idleImageName	File name of an optional idle image file to be displayed on the supported device (Ingenico Tetra devices only)
idlePrompt	An optional idle prompt to be displayed on the PIN device. When not provided the triPOS logo or a custom idle Image will be displayed instead.
language	The default language used for prompts on the payment device.
marketCode	The market code specific to the types of transactions that will be submitted. • Values: Default, AutoRental, DirectMarketing, Ecommerce, FoodRestaurant, HotelLodging, Petroleum, Retail, QSR

	• This value can be overridden on a per-request basis by setting the marketCode value in the configuration section of applicable requests.
printReceiptChoice	UK only. When using the terminal-ui module configures the SDKs use of the integrated DX8000 printer. Values: Never (Never print a receipt), Merchant (Only print merchant receipt), Ask (Always print merchant receipt and ask for customer receipt to print)
verifySignatureEnabled	UK only. When true and the Payment Platform requests a confirmation of the cardholder signature the SDK will raise an event via the DeviceInteractionListener.

Device Configuration

bluetoothConfiguration	The configuration object used to get and set the Bluetooth identifier for the POI device. bluetoothIdentifier – An optional Bluetooth identifier to specify the Bluetooth device to establish connection with.
contactlessAllowed	The Boolean value that determines whether EMV contactless transactions are allowed.
contactlessDetectionTime	The value used to set the contactless detection time in seconds. In case of MOBY device, maximum allowed time is 2.5 seconds and above this limit, value will be reset to 2.5 seconds.
deviceType	The enumeration for supported device and connection type. • BBPosDevice - BBPOS

	◦ Used for BBPos Chipper 2XBT and 3XBT devices • IngenicoAxium - Ingenico Axium ◦ Used for the DX8000 and EX8000 • IngenicoRbaBluetooth - Ingenico RBA, Bluetooth connection ◦ Used for iSMP4/iSMP4n (no barcode support) • IngenicoRbaTcpIp - Ingenico RBA, TCP/IP connection ◦ Used for iSMP4/iSMP4n (no barcode support), iPP350 • IngenicoUppBluetooth - Ingenico UPP, PCL Bluetooth connection ◦ Used for Link 2500, Move 5000 • IngenicoUppTcpIp - Ingenico UPP, TCP/IP ◦ Used for Lane 3000, Link 2500, Move 5000, Lane 3600 • IngenicoUppUsb - Ingenico UPP, USB connection ◦ Used for Lane 3000, Link 2500, Lane 3600 • Null - Null device ◦ Used for development testing of devices with a PIN pad or display. • NullDeviceWithNoPinpadDisplay - Null device with no PIN pad or display ◦ Used for development testing of devices which use the deviceInteractionListener • IngenicoRuaBluetooth - Ingenico RUA (Moby) Bluetooth capable devices ◦ Used for Moby5500 and Moby5500M (CL1 and CL3) • IngenicoRuaUSB - Ingenico RUA (Moby) devices connected through USB ◦ Used for Moby5500 and Moby5500M (CL1 and CL3)

identifier	The Bluetooth identifier will depend on the device and what be used to specify the correct device. • Ingenico - The format of the Ingenico device identifier is "xxx-yyyyyyyy", where "xxx" is the device type and "yyyyyyyy" is the last 8 digits of the serial number found on the label on the back of the device. See examples of the currently supported Bluetooth devices below: ▪ iSMP4: iSMP4-12345678
keyedEntryAllowed	The Boolean value that determines whether keyed entry transactions are allowed.
rebootTimeHour	The value used to set the reboot time hour. • This value should be 0-23.
rebootTimeMinutes	The value used to set the reboot time minutes. • This value should be 0-59.
tcpIpConfiguration	The configuration object to manage the TCP/IP properties. • ipAddress: The IPv4 address of the POI device to connect to. • Port: The 16-bit TCP port number to which the PIN pad is listening for a connection. This value is configured on the PIN pad.
terminalId	The unique identifier of the terminal / PIN pad for *Express*
terminalType	Indicates what function / type of PIN pad is being used • Values: Unknown, PointOfSale, Ecommerce, Moto, FuelPump, ATM, Voice, Mobile
p2peKeySlotIndex	The index of the key used for card data encryption • Note: this config is only used for Moby 5500

wifiRoamingEnabled	The Boolean value that determines whether Wi-Fi Roaming is enabled(true) or disabled(false) • Note: this config is only used for Link 2500 and Move 5000
heartbeatEnabled	The Boolean value that enables\disables the heartbeat to manage device connection. When enabled, triPOS SDK will handle reconnection to the device in case of any connection failures. *Supported on Ingenico Tetra and Ingenico Moby devices.*
devicePool	The Device pool can have one or more Ingenico Moby devices. Each device in the pool needs a type (IngenicoRuaUsb), USB Configuration that includes the serial number, and a description of the device. *Supports only Ingenico Moby devices on USB.*
sleepTimeoutSeconds	The value used to set the sleep time in seconds. • This value should be between 1-3600. A value of 0 disables sleep mode, which is not recommended. Note: this config is only used for Moby 5500
barcodeReaderEnabled	The Boolean value that determines whether QR Code and Barcode scanning transactions are allowed. Note: this config is only used for Axium Devices (DX8000, DX4000, EX8000)

Host Configuration

acceptorId	For Express: Your *Express* acceptor ID which is also your Merchant ID. For A4P: Your Terminal ID (TID)

accountId	For Express: Your *Express* account ID For A4P: Your Merchant ID (MID)
accountToken	Your *Express* or *A4P* account token
applicationId	Unique application identifier.
applicationVersion	The value used to set the application version
applicationName	The value used to set the application name.
expressRequestTimeout (Deprecated)	Deprecated with 4.4.0. No longer supported instead use hostRequestTimeout.
hostRequestTimeout	Sets the maximum number of seconds to wait when communicating with a remote host. Value will be coerced to between 35 and 75 seconds.
paymentProcessor	The enumeration of the payment processor to be used which allows the triPOS® Mobile SDK to handle processor specific implementation mainly for EMV. • Values: Worldpay, TSYS, WorldpayUK, WorldpayCanada, WorldpayROI, Global, ChasePaymentech and FirstData
reversalRetryLimit	If a communication error occurs among triPOS®, the processor and the host, triPOS® will try to reverse the transaction to allow you to resend it while avoiding duplicates. This value limits how many times triPOS® tries to reverse the transaction before returning to the client application. The default value is 1, and the maximum value is 4.
storeCardId	Optional store card ID required to process store card transactions.
storeCardPassword	Optional store card password required to process store card transactions.
vaultId	Optional vault ID used in token creation.

Payment Processors

The payment processor determines which Worldpay Integrated Payments Platform will be used to process the payments as well as setting the country of the terminal.

Payment Processor	Locale			
	US	Canada	UK	ROI
Worldpay	✓			
TSYS	✓			
WorldpayUK			✓	
WorldpayCanada		✓		
WorldpayROI				✓
ChasePaymentech	✓			
Global	✓			
FirstData	✓			

Feature	Payment Processor			
	Worldpay	TSYS, ChasePaymentech, Global, First Data	WorldpayUK	WorldpayCanada
Sales	✓	✓	✓	✓
Refunds	✓	✓	✓	✓
Authorizations	✓	✓	✓	✓
Gift cards	✓	✓		

EBT	✓	✓		
Store and forward	✓	✓		✓
Quick chip	✓	✓		

☞ Note: The following functionality is not currently support with the WorldpayUK processor.

- Health check
- Reversal
- Return
- Hosted payments
- Quick chip
- Pre-read
- Surcharging
- Convenience Fee
- HSA
- Token (Create token, Sale with token, refund with token, etc.)
- Gift card transactions (Activate, Balance enq., Reload, Close, etc.)
- EBT voucher transaction
- Manually forward (store and forward transactions)
- DCC
- Partial approval

Store and Forward Configuration

isStoringTransactionsAllowed	The Boolean value that determines whether transactions are allowed to be stored.
numberOfDaysToRetainProcessedTransactions	An integer value that represents the number of days to retain processed transactions. The default value is set to the maximum of 120 days, and the minimum value is 1 day.

shouldTransactionsBeAutomaticallyForwarded	The Boolean value that determines whether transactions should be automatically forwarded while the SDK is initialized and can reach a connection with *Express*.
transactionAmountLimit	An integer value that represents the maximum dollar amount of a transaction that can be stored when the SDK is initialized but cannot reach a connection with *Express*.
TransactionStoringMode	The enumeration value which determines the store and forward mode. • Disabled – Store and forward is disabled. • Manual – The SDK will attempt to store the transaction no matter the status of connection with Express. • Auto – When Express cannot be reached, the SDK will attempt to store the transaction based on configuration.
unprocessedTotalAmountLimit	When storing, triPOS® will decline any transaction that causes the sum of all unprocessed transactions to exceed this maximum amount.

Transaction Configuration

addressVerificationCondition	The enumeration value which determines if the transaction should require an address verification. • Never - The transaction flow will not prompt for an address verification.

	• Always - The transaction flow will always prompt for an address verification. • Keyed - Only keyed entry transactions will require an address verification.
aidPreference	The enumeration value which determines which Application Identifiers (AIDs) will be returned based on the supported applications on the card. • NoPreference - Returns all supported AIDs on the card. • USCommonDebit - Returns all USCommonDebit and Global AIDs supported on the card. • Global - Returns all Global AIDs supported on the card. [Change requires the SDK to be reinitialized]
amountConfirmationEnabled	The Boolean value that determines whether confirmation of the total amount for the transaction happens during the transaction.
cashbackAllowed	The Boolean value that determines whether cashback is allowed.
cashbackEntryAllowed	The Boolean value that determines whether cashback entry is allowed. This value only has effect when cashbackAllowed is set to true.
cashbackEntryIncrement	The integer value which sets the cashback entry increment allowed.
cashbackEntryMaximum	The integer value that determines the maximum cashback entry allowed.

cashbackOptions	The BigDecimal array that represents the cashback options presented to the user.
convenienceFeeConfirmationEnabled	The Boolean value that determines whether confirmation of the convenience fee happens during the transaction.
currencyCode	The currency code that will be used for all transactions. • Values: USD (US Dollars), CAD (Canadian Dollars), GBP (Great British Pounds).
debitAllowed	The Boolean value that determines whether debit transactions are allowed (on supported devices). The credit functionality of check cards will still work if this is set to false.
duplicateTransactionAllowed	The Boolean value that determines whether duplicate transactions should be approved or declined.
emvAllowed	The Boolean value that determines whether EMV transactions are allowed. The default value is false. [SDK needs to be reinitialized if changed]
emvCardInputFailureRetriesEnabled (Deprecated)	The Boolean value that determines whether an EMV card input can be retried before entering fallback.
cardInputRetries (Deprecated)	Deprecated with 4.3.0. No longer supported. Instead use the swipeAttempts, contactAttempts and contactlessAttempts fields to specify the number of card input attempts.
giftCardAllowed	The Boolean value that determines whether gift card transactions can be processed.

isCustomAidSelectionEnabled	The Boolean value that represents whether custom aid selection is enabled. • If set to **true,** the AID selections will appear as "CREDIT" or "DEBIT" rather than by Application Label Names. • If there are more than two AIDs returned, then all AID Labels will be displayed. *[Not supported on all devices]*
isDynamicCurrencyConversionEnabled	The Boolean value that determines whether dynamic currency conversion is enabled.
isEbtCashBenefitsAllowed	The Boolean value that determines whether EBT cash benefits is allowed.
isEbtFoodStampsAllowed	The Boolean value that determines whether EBT food stamps is allowed.
isHealthcareSupported	The Boolean value that determines whether HSA/FSA healthcare qualified are supported transactions. The default value is **false**. • If set to **true,** your application must support transactions for healthcare qualified goods. Visit the SIG-IS site https://www.sig-is.org.
partialApprovalAllowed	The Boolean value that determines whether partial approvals are allowed. If set to **true**, the business application is equipped to handle partial approvals, a card brand requirement in card-present merchant environments.
preReadQuickChipPlaceHolderAmount	BigDecimal value used to set the pre-read quick chip place holder amount. The default value is $1

quickChipAllowed	The Boolean value that determines whether the transaction should be run as quickchip. *[Not supported on all devices or processors]*
securityCodePromptEnabled	The Boolean value that determines whether a keyed transaction should prompt for the card security code (CSC).
shouldConfirmSurchargeAmount	The Boolean value that determines whether the transaction flow should prompt the user to confirm the surcharge amount.
tipAllowed	The Boolean value that determines whether the transaction should prompt for a tip.
tipEntryAllowed	The Boolean value that determines whether a tip entry is allowed.

tipOptions	A BigDecimal array of tip options.
	
	• Values: Dependent on the tipSelectionType
	  o Example: tipSelectionType.Amount: "1, 2, 3, 4" ($1, $2, $3, $4)
	  o Example: tipSelectionType.Percentage "0.05, 0.1, 0.15, 0.2" (5%, 10%, 15%, 20%)

tipSelectionType	The enumeration for the tip selection type. • Amount - Tip represented as a dollar amount. • Percentage - Tip is represented as a percentage amount.
contactAttempts	The maximum number of contact attempts to read the card during a payment transaction. The default value is 3. The allowed values are between 1 and 3.

contactlessAttempts	The maximum number of contactless attempts to read the card during a payment transaction. The default value is 3. The allowed values are between 1 and 3.
swipeAttempts	The number of swipe attempts (including fallback in the case of chip read failure) to read the card during a payment transaction. The default value is 3. The allowed values are between 1 and 3.
preReadDataLifetimeSeconds	The duration, in seconds, that pre-read card data will remain valid. The default value is 600. The allowed values are between 30 and 900.
isFeeAssistEnabled	The Boolean value that determines whether Fee Assist (Hosted Credit Surcharge) is enabled. When true, the SDK will perform surcharge inquiries with Express before processing credit card transactions to determine applicable surcharge amounts. Default value is false. Note: Fee Assist is only available in the United States.
proceedWithoutSurchargeOnError	The Boolean value that determines whether transactions should proceed without surcharge when a surcharge inquiry fails in Auto Store & Forward mode. When true, transactions will be stored with the original amount if the surcharge inquiry fails. When false, the transaction will be terminated. Default value is false.

Card Input Retries

Integrators can use the transaction configuration fields(“contactlessAttempts”, “contactAttempts” and “swipeAttempts”) to control the number of attempts to read the card during a payment transaction. This includes the first attempt and the subsequent retries. This is supported on Ingenico Tetra, Ingenico Axium and Ingenico Moby devices.

When all contactless attempts are exhausted or if the contactless card cannot be read (when there are no matching applications), contactless will be disabled and the reader would prompt for an Insert or a swipe.

Similarly, when the contact attempts are exhausted or if the contact read cannot be performed (not matching applications or empty candidate list) both contact and contactless interfaces will be disabled and configured number of "swipeAttempts" will be provided to perform a fallback read. The transaction will be terminated if all attempts to read the card fails.

Transaction Requests

- AuthorizationCompletionRequest
- AuthorizationRequest
- AuthorizationWithTokenRequest
- CreateTokenRequest
- CreateTokenWithTransactionIdRequest
- CreditCardAdjustmentRequest
- EbtBalanceInquiryRequest
- EbtVoucherRequest
- GiftCardActivateRequest
- GiftCardBalanceInquiryRequest
- GiftCardBalanceTransferRequest
- GiftCardCloseRequest
- GiftCardReloadRequest
- GiftCardUnloadRequest
- HealthCheckRequest
- IncrementalAuthorizationRequest
- ManuallyForwardRequest
- RefundRequest
- RefundWithTokenRequest
- ReturnRequest
- ReversalRequest

- SaleRequest
- SaleWithTokenRequest
- TransactionSetupRequest
- VoidRequest

Many to Many (Multiple Device Pool)

Overview

The feature allows the business application to define a device pool under the device configuration. The device pool may have one or more Moby devices connected via USB. The new initialize device pool function will connect, initialize, configure, and prepare each device in the device pool sequentially. The connection is dropped once the device setup is complete. When all the devices in the pool are successfully initialized, a session may be started with a single device from the device pool. Starting a session establishes a connection with the selected device. During an active session, a transaction like a sale or device operation like reading a card may be performed. A session can be closed when the active connection is no longer required. Closing the session disconnects the payment device from the Android host device and returns the device to the list of available devices within the pool.

Device Pool

The Device pool may have one or more (maximum of 10 devices) Ingenico Moby 5500 devices (DeviceType. IngenicoRuaUsb). Each device in the pool needs the configuration values below.

Definition

```
public class UsbConfiguration extends ConfigurationValidation
{
    Private String serialNumber;
}
```

USB

serialNumber	The serial number of the USB device.

Device configuration

```
public class DeviceConnectionInfo extends ConfigurationValidation
{
    private String description;

    private DeviceType deviceType;

    private UsbConfiguration usbConfiguration;

    private Integer p2peKeySlotIndex;
}
```

description	The application supplied description for this device. This value is passed back with any initialization event or error.
deviceType	The type of this device. NOTE: Currently only DeviceType. IngenicoRuaUsb is supported.
usbConfiguration	The USB configuration for this device.
P2peKeySlotIndex	The index of the key used for card data encryption. Default value is 0

Usage

```
ArrayList<DeviceConnectionInfo> devicePool = new ArrayList<DeviceConnectionInfo>();

//Add the first device
DeviceConnectionInfo *device1 = new DeviceConnectionInfo();
device1.setDescription("Lane 1"); // User description for the lane.
device1.setDeviceType(DeviceType.IngenicoRuaUsb); // Only DeviceType.IngenicoRuaUsb
device1.setP2peKeySlotIndex(0); // Set index of the key to be used.

UsbConfiguration usbConfiguration1 = new UsbConfiguration();
usbConfiguration1.setSerialNumber("123456");
device1.setUsbConfiguration(usbConfiguration1);
devicePool add(device1);
             .
             .
             .
//Add the Nth device
DeviceConnectionInfo deviceN = new DeviceConnectionInfo();
deviceN. setDescription("Lane N"); // User description for the lane.
deviceN.setP2peKeySlotIndex(0)
deviceN.setDeviceType(DeviceType.IngenicoRuaUsb); // Only DeviceType. IngenicoRuaUsb
UsbConfiguration usbConfigurationN = new UsbConfiguration();
usbConfiguration1.setSerialNumber("987654");
deviceN.setUsbConfiguration(usbConfigurationN);
devicePool add(deviceN);

//Assign the device pool
configuration.deviceConfiguration.setDevicePool(devicePool);
```

Device Pool Initialization

Calling **initializeDevicePool** only once to initialize the **triPOS**® Mobile SDK to connect and initialize all the devices in the defined pool. The device pool needs to be initialized before an active session may be established with any device. The **triPOS**® Mobile SDK sequentially connects to each device in the pool and sets up the configuration. For each device, if the connection and configuration is successful, a completion callback is sent with the details of the device including the identifier. An error callback is made with a detailed error message if an error in connection or the configuration occurs. Once all the devices in the pool are configured, a completion message is sent with a summary of all the devices in the pool along with the status of the configuration. After completing the setup of each device, the **triPOS**® Mobile SDK disconnects from the devices.

When the device pool is initialized, each instance attempts to establish connection and configure the payment devices. The SDK reconfigures the device if there are any changes to the core configuration. When there are no changes, the device is not reconfigured during the device pool initialization.

- ☞ Note: The **triPOS**® Mobile SDK will not try to reconnect to the device nor reattempt the connection in case of a connection failure during the device pool setup. An appropriate error is returned.
- ☞ Note: Heartbeats are not sent to the device during the device pool initialization. On any error during the connection or configuration, the SDK moves on to the next device in the pool.
- ☞ Note: A session cannot be started while the device pool initialization is in progress.

initializeDevicePool

Prototype

```
public void initializeDevicePool(Context applicationContext, Configuration configuration, DevicePoolSetupListener devicePoolSetupListener) throws Exception
```

```
public void initializeDevicePool(Context applicationContext, Configuration configuration, DevicePoolSetupListener devicePoolSetupListener, OTAUpdateListener otaUpdateListener) throws Exception
```

Parameters

applicationContext	The application context.
configuration	The configuration for this instance of the triPOS® ® Mobile SDK. This includes the device pool.
devicePoolSetupListener	Class implementing the DevicePoolSetupListener interface to receive notifications during initialization.
otaUpdateListener	The OTA update listener.

DevicePoolSetupListener

The triPOS® Mobile SDK uses the callbacks to notify the business application about various events during the device pool initialization. The DevicePoolSetupListener interface defines methods the business application needs to implement to receive these notifications and events.

onDeviceSetupCompleted

The onDeviceSetupCompleted method is called when a device from the device pool has been successfully configured.

Prototype

void onDeviceSetupCompleted(String description, String model, String serialNumber, String firmwareVersion, String batteryLevel, String deviceDescription, String address)

Parameters

deviceInformation	Description of the connected device
model	Device model information
serialNumber	Serial number of the connected device
firmwareVersion	Version of the firmware on the device
batteryLevel	Not applicable for Ingenico Tetra devices. Not used.
deviceDescription	Device identifier from the device pool configuration.
address	Port identifier to which the connection was established.

onDeviceSetupError

The onDeviceSetupError method is called if there is an error during the setup of a device from the device pool.

Prototype

```
void onDeviceSetupError(Exception exception, String description, String address)
```

Parameters

error	Pointer to an error object with details of the issue during configuration
description	Device identifier from the device pool configuration.
address	If possible, the port identifier to which the device connected.

onDevicePoolSetupCompleted

The onDevicePoolSetupCompleted method is called when the device pool initialization has been completed.

Prototype

```
void onDevicePoolSetupCompleted(ArrayList<DeviceSetupResult> deviceResults)
```

Parameters

deviceResults	Summary of the device pool initialization. Includes the list of devices from the device pool along with the status of the setup and an error object if the device setup failed.

DeviceSetupResult

didConfigureSuccessfully	Boolean value indicating whether the device connected successfully.
exception	Exception indicating the reason the device did not connect successfully.
deviceInformation	Description of the connected device.
model	Model of the connected device.
serialNumber	Serial number of the connected device.
firmwareVersion	Firmware version of the connected device.
batteryLevel	The battery level of the connected device. The property is only used if the device is battery powered. Depending on the device, this property may contain the percentage of the battery charge or may contain a description of the battery level (e.g. "Low", "Critical"…).
deviceDescription	Device identifier from the device pool configuration,
address	If possible, the port identifier to which the device connected.

Start Session

A new session is established using the startSession method by providing the device identifier. A session can only be started after the device pool has been initialized, and with an available device from the pool. During the start session operation, the SDK establishes a connection with the payment device. The payment device is ready to process any transaction if previously configured successfully. If the device does not have the required configuration, the SDK prompts for confirmation to continue with the configuration before the session can be established. Upon successful connection, the device information along with the device identifier is sent back to the business application. An error response is returned if an error occurs during the connection or configuration of the device.

The triPOS® Mobile iOS SDK sends heartbeats to the payment device at regular intervals to maintain the active session and attempt to reconnect in case of any connection failures.

startSession

Prototype

```
void startSession(Context applicationContext, DeviceConnectionInfo deviceConnectionInfo,
StartSessionListener startSessionListener) throws Exception;
```

Parameters

applicationContext	The application context.
deviceIdentifier	The identifier supplied during initialization of the requested device.
startSessionListener	Class implementing the StartSessionListener interface to receive notifications during while starting the session.

onConnected

The onConnected method is used when a session has been successfully established.

Prototype

```
void onConnected(String description, String model, String serialNumber, String firmwareVersion,
String batteryLevel, String deviceDescription, String address)
```

Parameters

description	Description of the connected device.
model	Device model information.
serialNumber	Serial number of the connected device.
firmwareVersion	Firmware version of the connected device.
batteryLevel	The battery level of the connected device. The property is only used if the device is battery powered. Depending on the device, this property may contain the percentage of the battery charge or may contain a description of the battery level (e.g. "Low", "Critical"…).
deviceDescription	Device identifier from the device pool configuration.
address	Port identifier to which the device connected.

onError

The onError method is used when there are errors during connection or configuration during a start session activity.

Prototype

```
void onError(Exception exception, String description, String address)
```

exception	Pointer to an error object with details of the issue during configuration
description	Description from the device pool of the device generating the error.
address	If available, the port identifier to which the device connected.

Confirm configuration on Start Session

If triPOS® Mobile iOS SDK determines the payment device does not have the required configuration at time of starting a new session, the onRequestConfigureDeviceOnStartSession callback is used to ask for confirmation to proceed with the configuration Upon confirmation, the configuration is pushed to the device and the session is established once complete. If the confirmation is declined, the start session returns an error. The business application must respond using the callback passed back in this delegate method.

onRequestConfigureDeviceOnStartSession

Prototype

void onReqeustConfigureDeviceOnStartSession(DisplayTextIdentifiers displayTextIdentifier, String confirmationMessage, StartSessionConfirmationListener startSessionConfirmationListener)

Parameters

displayTextIdentifier	Text identifier of prompt to request permission to configure a device at session start if necessary.
confirmationMessage	Message to be shown to the user to confirm configuration should continue or not.
startSessionConfirmationListener	Class that implements the StartSessionConfirmationListener to receive the result of the confirmation.

StartSessionConfirmationListener

The StartSessionConfirmationListener interface defines the method(s) needed to be implemented to handle the result of onReqeustConfigureDeviceOnStartSession.

```
public class StartSessionConfirmationListener extends EventsListener
{
    void setDeviceConfigurationConfirmation(boolean confirm);
}
```

Close Session

When an active connection with the device is no longer required, the business application may request to close the active session using the closeSession method. A session can be closed only if there are no ongoing activities or transactions. During a close session, the triPOS® Mobile SDK disconnects the device and stops sending heartbeats. A successful close session ensures the device returns the device to the pool and available to start new session.

closeSession

Prototype

public void closeSession(CloseSessionListener closeSessionListener) throws Exception

Parameters

CloseSessionListener	Listener to receive notifications with results of the close session request.

onSessionClosed

The callback method used when a session has been successfully closed.

onSessionClosed()

onError

Used when there are any errors while closing the device session.

onError(exception: DeviceDisconnectionException)

DeviceDisconnectionException	Details of any error while closing an active session with the device.

Dependencies

triPOS® leverages several libraries in the form of .aar files. These files need to be added as implementation dependencies in the application build.gradle based on which devices and connection types will be utilized in the application. In the sample app packaged with the SDK, these files can be found in the sampleapp\libs folder.

triPOS® Mobile SDK

The triposmobilesdk-release.aar file must be added for all device types and connection types.

- implementation (name:'triposmobilesdk-release', ext:'aar')

Apache log4j Libraries

The log4j-api and log4j-core libraries are required to recover the triPOS SDK logs from path: Android/data/integrator_app_packagename/files/fis/tripossdk/log/*.log.

- implementation 'org.apache.logging.log4j:log4j-api:2.17.1'
- implementation 'org.apache.logging.log4j:log4j-core:2.17.1'

RBA_SDK

The rba_sdk.aar file must be added for Ingenico Devices: (iSMP4/iSMP4n, iPP 350, Link 2500, Lane 3000, Move 5000, DX8000 and EX8000). The latest version of RBA SDK is 5.33.9.

- implementation (name:'rba_sdk', ext:'aar')

PCL Libraries

The iPclBridge.aar, PclServiceLib_2.21.02.aar, and PclUtilities_2.21.02.aar files must be added for IngenicoUPP devices connecting via PCL Bluetooth (Link 2500, Move 5000).

- implementation (name:'iPclBridge', ext:'aar')
- implementation (name:'PclServiceLib_2.21.02', ext:'aar')
- implementation (name:'PclUtilities_2.21.02', ext:'aar')

Ingenico Axium Retail Core (ARC) Libraries

The retail-types-release- 22.01.06.01-0010.aar and ux-server-release- 22.01.06.01-0010.aar files need to be added for use with the Ingenico Axium devices (EX8000/DX8000).

- implementation (name:'retail-types-release- 22.01.06.01-0010, ext:'aar')
- implementation (name:'ux-server-release- 22.01.06.01-0010, ext:'aar')

RUA SDK

The RUA SDK is leveraged by the triPOS SDK for supporting communication with Ingenico RUA devices (Moby 5500 and Moby5500M)

- roamreaderunifiedapi-2.5.3.100-release.aar

Other Libraries

- implementation 'io.realm:realm-gradle-plugin:10.15.1'

- implementation 'androidx.appcompat:appcompat:1.7.0'
- implementation 'androidx.core:core-ktx:1.7.0'
- implementation 'androidx.work:work-runtime:2.10.0'
- implementation 'androidx.work:work-runtime-ktx:2.10.0'
- implementation 'com.google.dagger:dagger:2.45'
- implementation 'com.google.code.gson:gson:2.13.1'
- implementation 'com.squareup.retrofit2:retrofit:2.9.0'
- implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
- implementation 'com.squareup.okhttp3:logging-interceptor:4.10.0'
- implementation 'com.squareup.okhttp3:okhttp-tls:4.10.0'
- implementation 'commons-io:commons-io:2.11.0'
- implementation 'commons-validator:commons-validator:1.7'
- implementation 'org.apache.commons:commons-compress:1.21'
- implementation 'org.apache.commons:commons-lang3:3.12.0'
- implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.7.0'
- implementation 'org.simpleframework:simple-xml:2.7.1'
- implementation 'com.journeyapps:zxing-android-embedded:4.3.0'
- implementation 'com.google.zxing:core:3.5.3'

Callback Listeners

Various listeners can be leveraged for different purposes while integrating with the triPOS SDK including obtaining information on device connection\disconnection, firmware updates, Bluetooth pairing confirmation, status of various transactions.

DeviceConnectionListener

POI device connection callbacks.

Method	Description	Parameters
onConnected	Called when the POI device connects.	Device - The device that connected. Device is a base class so this parameter may be one of several types. Description - The device description.

		Model - The device model. serialNumber - The device serial number.
onDisconnected	Called when the POI device disconnects	Device - The device that disconnected. Device is a base class so this parameter may be one of several types.
onError	Called when the POI device returns an error	Exception - The exception that occurred
onBatteryLow	Called when the POI device returns a low battery warning. (Supported on BBPos and Moby)	
onWarning	Called when the POI device returns a warning	
onConfirmPairing	Called with BT pairing needs to be confirmed (Applicable for Moby BT connections)	ledSequence – LED sequence displayed on the device to confirm device pairing. deviceName – name of the device being connected. confirmPairingListener – callback listener used to send the pairing confirmation or cancellation back to the SDK.

OTAUpdateListener

The OTAUpdateListener provides over-the-air firmware and configuration updates for BBPos devices.

Method	Description	Parameters
onOverTheAirUpdateStarted	Called to inform the delegate that an over the air device update has started.	otaUpdateType – The type of update (config or firmware) deviceInfo – Properties of the connected device.
onOverTheAirUpdateFinished	Called to inform the delegate on the progress of an over the air device update.	otaUpdateType - T The type of update (config or firmware) deviceInfo - Properties of the connected device.
onOverTheAirUpdateProgress	Called to inform the delegate on the progress of an over the air device update.	otaUpdateType - The type of update (config or firmware) deviceInfo - Hashtable of connected device for which the update is being done.

InitializationProgressStatusListener

Method	Description	Parameters
onInitializationStatusChanged	Called when the initialization progress status changes.	Status – Current step\status. progressValue – progress percentage (where applicable) of current step. initializationProgressDescription – additional information related to the status.

DeviceInteractionListener

Interface for UI Display and User Input request notifications for no-PIN pad and no-display devices.

Method	Description	Parameters
onAmountConfirmation	Called when amount confirmation is needed for use with no PIN-pad devices.	amountConfirmationType - The description of the amount being confirmed, total amount, convenience fee, etc. amount - The amount being confirmed. confirmAmountListener - The listener to be used to send back the confirmation result.
onChoiceSelections	Called when user is presented with a list of choice - needed for use with no PIN-pad devices.	Choices - The list of choices to select from. selectionType - What the list of choices is for (i.e.. payment type, tip amount, tip confirmation) title - Title of the dialog. selectChoiceListener - The listener that is to be notified with the array index of the selected choice.

onNumericInput	Used when user is presented with a numeric input field like a zip code or a custom tip amount	numericInputType - The type of numeric input requested - i.e. Tip, etc. numericInputListener - The listener that is to be notified with the numeric input.
onSelectApplication	Used when application selection for EMV card is required - needed for use with no PIN-pad devices	applications - The list of applications to select from. selectApplicationListener - The listener that is to be notified with the array index of the selected application.
onPromptUserForCard	Called when the user needs to be prompted to present a card.	prompt - The text to be used when prompting user for card input
onPromptUserForCardWithBarcode	Called when the user needs to be prompted for card input with barcode scanning support.	prompt - The text to be used when prompting user for card input. barcodePrompt- The text to be used when prompting user to scan a barcode.
onDisplayText	Called to display informative text to the user that requires no response. Used with devices with no display of its own.	text - The text to be displayed to the user.

onWait	Called to display informative text to the user while waiting for a process to complete	message - The text to be displayed to use while waiting
onRemoveCard	Called to inform the user the EMV card inserted needs to be removed.	
onCardRemoved	Called to notify the EMV card inserted has been removed. Used to dismiss any messages shown to the user to remove card.	
onVerifySignature	UK only. Called when the merchant is required to check the cardholder signature.	saleResponse – Information about the current sale. verifySignatureListener – callback used to return the merchants approve/decline response.

setDeviceInteractionListener	To set the deviceinteractionlistener globally	globalDeviceInteractionListener: The listener to handle global device interaction events.

Response Listeners

The different methods in the device interaction listener also provides callback listeners through which the user choice or the user response can be sent back to the triPOS SDK.

Method	Description	Parameters
NumericInputListener	User together with the OnNumericInput method of device interaction listener to pass back the user input value.	enterNumericInput(String numericInput): A string containing the user input
SelectChoiceListener	Used with onChoiceInput method of the device interaction listener to passback the user selection among the available choices. The	selectChoice(int selectedChoice): an int that represents the selected choice

	index of the user selection needs to be returned.	
ConfirmAmountListener	Used with onAmountConfirmation to send back the user confirmation response.	confirmAmount(Boolean amountConfirmed): Boolean indicating if the amount was confirmed.
ConfirmPairingListener	Used with onConfirmPairing method of device connection listener to return the user response to BT pairing confirmation.	Confirm Pairing: Used to confirm the Bluetooth pairing with a Moby device during the triPOS SDK initialization. Cancel Pairing: Used to cancel the Bluetooth pairing of a Moby Bluetooth device during the initialization.

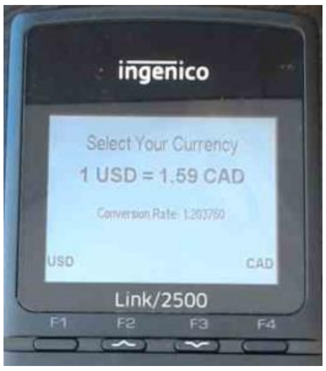
Express Hosted Payments

Express Hosted Payments is a PCI certified mechanism for keyed card entry, specifically for devices that do not have the capability (non-display, non-keypad). triPOS® Mobile SDK integrates this Express feature just as any other integrator would. The integrating application is responsible for collecting all the non-sensitive data needed to perform a payment transaction.

triPOS® Mobile SDK provides APIs for sales and authorizations to make it easy and convenient for an integrator to incorporate Express Hosted Payments into the payment application. These APIs provide mechanisms to setup the Express transaction and returns all the data necessary to process the transaction using Hosted Payments in an embedded web browser from the integrator's application. Once the transaction is completed, triPOS® Mobile SDK, provides additional APIs to parse the Hosted Payments response URL and return a familiar sale response or authorization response for the integrator's use. In addition, triPOS® Mobile SDK provide samples for this in the included Sample App.

Dynamic Currency Conversion

The triPOS® Mobile SDK leverages the Express gateway to support Dynamic Currency Conversion (DCC) functionality which allows the cardholder to choose in which currency to pay. Prior to processing the transaction, if the card is eligible for DCC, the terminal presents the transaction amount in the merchant's currency and the default currency for the card, the conversion rate, and any fees/markups related to the conversion. If the cardholder chooses their currency, the rate is locked in, and the transaction completes. If the cardholder chooses the merchant currency, the transaction completes normally and the transaction is converted later by the card issuer. Dynamic Currency Conversion (DCC) is enabled or disabled with the isDynamicCurrencyConversionEnabled flag in the TransactionConfiguration class.



BIN Lookup

Enhanced BIN Query and Payment Type Selection

Enhanced BIN Query is supported on the *Express* platform. This feature is not available with the *Access 4 POS* (*A4P*) platform.

The payment type selections of swipe transactions are determined by the Express Enhanced BIN Query result. If there is only one payment type or the result matches only one configured payment type, it will automatically be selected. If the Enhanced BIN query returns a result, then any payment type configured that is enabled in the BIN will be options for selection. If the Enhanced BIN Query fails to return a result, then all payment types configured will be options for selection.

Since test cards typically do not show up in production BIN tables, triPOS® Mobile SDK has some built in responses that may be used for testing purposes only. The below transaction amounts will prompt for the corresponding payment types.

- ☛ Note: These built-in responses are only available when triPOS® Mobile SDK is operating in test/certification mode.

Enhanced BIN Query trigger values

Transaction Amount	Card Types Enabled
11.11	Credit only
11.21	Debit only
11.31	CheckCard, Debit
11.41	EBT only
11.51	Credit, GiftCard
11.61	Credit, Debit, EBT, Gift
0.60	Gift only
All other Values	All payment types enabled

Local BIN file

The triPOS Mobile SDK supports local BIN lookup for transactions such as Sale, Authorization, and Refunds by downloading a BIN file. This file is downloaded once every seven days, provided the SDK verifies network access and sufficient storage space. After the file is successfully downloaded and parsed, it is automatically deleted to free up resources.

BIN and Pre-Read

The triPOS Mobile SDK performs a BIN lookup using the local BIN file (if available) as part of a pre-read. The BIN information obtained from the BIN lookup is included with the pre-read response. This response includes the BIN value, and the card's capabilities, such as whether it is credit, debit or gift capable.

Surcharge and Convenience Fee

Surcharging and Convenience Fees are supported the *Express* platform. These features are not available with the *Access 4 POS* (*A4P*) platform.

Surcharge and convenience fees are ways for businesses to recoup some of the money they spend on processing fees. A surcharge is a fee you can add to every credit card purchase made by your customers. A convenience fee is a charge added when your customers make a purchase using a nonstandard payment type. Both surcharges and convenience fees are regulated by credit card companies and governments, so it is crucial to tell them apart if you plan on using one.

Standard Surcharge Implementation

The triPOS® Mobile SDK offers optional fields in the financial transaction request for both a fixed rate surcharge fee and a convenience fee. If there is an amount included in the request, it will be added to the Express request and incremented to the total amount. The triPOS® Mobile SDK also has configuration fields in the transaction configuration (convenienceFeeConfirmationEnabled, shouldConfirmSurchargeAmount) which will enable or disable a confirmation prompt for each fee.

Fee Assist (Hosted Credit Surcharge)

Fee Assist is a compliant credit surcharge product that allows merchants to add a fee to a transaction when paying with a credit card. When Fee Assist is enabled (via the isFeeAssistEnabled configuration flag), the SDK will:

1. Perform a surcharge inquiry - Before processing the transaction, the SDK calls Express to determine the applicable surcharge amount based on the card type and transaction details.

2. Use the Express-determined amount - The surcharge amount returned by Express takes precedence over any manually configured surcharge amount in the request. Any surcharge amount in the original request will be ignored.
3. Display confirmation prompt - If the surcharge amount is greater than zero and shouldConfirmSurchargeAmount is enabled, the cardholder will be prompted to confirm the surcharge amount.
4. Process with the total amount - If confirmed, the transaction proceeds with the original amount plus the surcharge. If declined, the transaction is cancelled.

Fee Assist and Store & Forward Modes

Fee Assist behavior varies based on the Store & Forward mode:

- Disabled Mode: Surcharge inquiry must succeed, or the transaction is terminated.
- Manual Mode: Fee Assist is not supported. A warning is sent indicating hosted surcharge is not supported when Manual storing is enabled. The transaction is stored with the original amount, ignoring any surcharge.
- Auto Mode:
  - If surcharge inquiry succeeds: Transaction proceeds normally with surcharge (if applicable)
  - If surcharge inquiry fails:
    - With proceedWithoutSurchargeOnError = true: Transaction is stored with original amount and a warning is sent
    - With proceedWithoutSurchargeOnError = false: Transaction is terminated

☞ Note: If Express cannot be reached and the local BIN lookup cannot determine if the card is credit only and capable of accepting a surcharge, a request with a surcharge amount will not be stored and the transaction will be cancelled.

☞ Note: Surcharging and Convenience fee may not be available for all device types. Surcharging for offline transactions is not offered for BBPOS Chipper devices.

☞ Note: Enhanced BIN lookup will not be performed when Fee Assist is enabled.

WiFi Roaming

Roaming allows a device to pick up another network access point when the device is moved away from the original access point. As the device is moved closer to the other access point (with the same IP address), the device connects to the new access point without breaking the existing connection.

Configuration in Device

WiFi roaming can be enabled for TCPIP/WIFI on the Ingenico Tetra device family for devices that support WiFi connectivity:

- Link 2500
- Move 5000

According to the Ingenico implementation guide, WiFi roaming is supported and can be configured using a custom LOCAL.DEVICE.COMMUNICATION.WIFI.PBT file. This configuration is downloaded to the terminal during the initialization process.

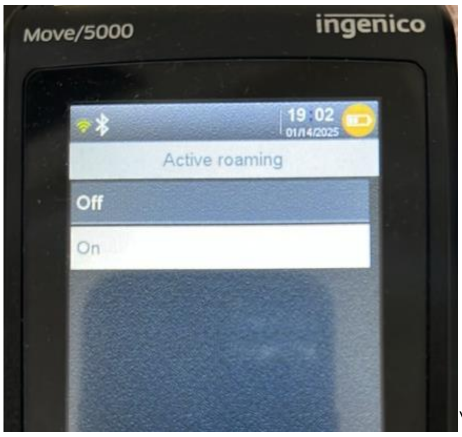
It's configurable to enable/disable through settings. If not set, then its default value remains 'Off'.

To access the WiFi roaming setting from the device:

- Enter key combination, 2634 -> F -> Control Panel -> Terminal Settings -> Comm means -> WiFI - > Advanced Options-> Active roaming – On/Off

Or

- press 0001 -> WiFi setting -> Active Roaming – On/Off



\

Configuration in SDK

The Wi-Fi roaming feature can be enabled or disabled via the wifiRoamingEnabled boolean variable in the Device Configuration of SDK. By default, wifiRoamingEnabled flag is set to FALSE.

Initialization

Upon SDK initialization, if the wifiRoamingEnabled is set to FALSE, the WIFI_ROAMING_OFF.TGZ file will be installed on the device, followed by a device reboot.
Conversely, if the wifiRoamingEnabled is set to TRUE, the WIFI_ROAMING_ON.TGZ file will be installed, and the device will subsequently reboot.

If the Wi-Fi roaming enabled flag remains unchanged from its previous state (i.e TRUE to TRUE or FALSE to FALSE), no installation or device reboot will occur.

QR Code and Barcode Scanning

triPOS SDK supports the scanning of QR Code and Barcode, enabling applications to initiate scans using either the front or rear camera. Upon a successful scan, the decoded information is returned to the application for further processing.

Device Compatibility

QR Code and Barcode scanning functionality is supported only on the following device models:

- Dx8000
- Dx4000

- EX8000

Workflow

1. SDK Initialization
   The triPOS SDK must be properly initialized prior to performing any scanning operations
2. Device Connection
   Once the SDK is initialized, the application must establish a connection with a device
3. Scanner Launch
   Once connected, the scanner interface can be launched. Users will be prompted to select either the front or rear camera for scanning purposes.
4. Data Capture and Return
   Upon successful scan of a QR Code or Barcode, the decoded data is captured and returned to the application.

Gift Card Transaction Support

The QR Code and Barcode Scanning feature supports the following gift cards transactions:

- Gift Card Activation
- Sale
- Balance Inquiry
- Reload
- Unload
- Balance Transfer
- Close

Configuration in SDK

The QR Code and Barcode Scanning feature can be enabled or disabled through barcodeReaderEnabled boolean variable within the SDK's Device Configuration. By default, this flag is set to FALSE.

Upon SDK initialization, if the barcodeReaderEnabled is set to FALSE, QR Code and Barcode scanning will remain unavailable. Conversely, if the barcodeReaderEnabled is set to TRUE, the scanning capability will be enabled.

Sample Application

triPOS® is packaged with a sample application. This application provides sample code for several functionalities that you may need to perform when integrating with the *triPOS® SDK*.

triPOSConfig.java

The triPOS.config file contains the host credentials, host configuration, device configuration, store and forward configuration and transaction-related settings. The triPOS.config file is used to customize individual merchant settings or establish a custom config build that is maintained from merchant to merchant and across triPOS®. Preservation of the custom triPOS.config settings is up to the integrating application.

setUpApplicationConfiguration

- The setUpApplicationConfiguration method is used to configure the application settings including the idle prompt/idle image and ApplicationMode.

```
private static void setupApplicationConfiguration()
    {
        ApplicationConfiguration appConfig = new ApplicationConfiguration();
        appConfig.setIdlePrompt("triPOS Samples");
        appConfig.setApplicationMode(ApplicationMode.TestCertification);
        sharedConfig.setApplicationConfiguration(appConfig);
    }
```

setUpHostConfiguration

- The setUpHostConfiguration method is used to set the credentials and other identifiable application properties for Integrated Payment System (*Express* or *A4P*).
- The hostConfiguration also contains credentials for specific features such as StoreCard and Token Creation functionality.

```
private static void setupHostConfiguration()
    {
        HostConfiguration hostConfig = new HostConfiguration();

        // Express test credentials, see http://www.elementps.com/Create-a-Test-Account
        credentials = new Credentials("1234567", "123456789ABCDEFG", "123");
        hostConfig.setAcceptorId(credentials.getAcceptorID());
        hostConfig.setAccountId(credentials.getAccountID());
        hostConfig.setAccountToken(credentials.getAccountToken());
        hostConfig.setStoreCardId("631522003");
        hostConfig.setStoreCardPassword("xyz");

        // Application config
        app = new Application("8414", "triPOS SampleApp", "0.0.0.0");
        hostConfig.setApplicationId(app.getApplicationID());
        hostConfig.setApplicationName(app.getApplicationName());
        hostConfig.setApplicationVersion(app.getApplicationVersion());
        hostConfig.setVaultId("174");

        sharedConfig.setHostConfiguration(hostConfig);
    }
```

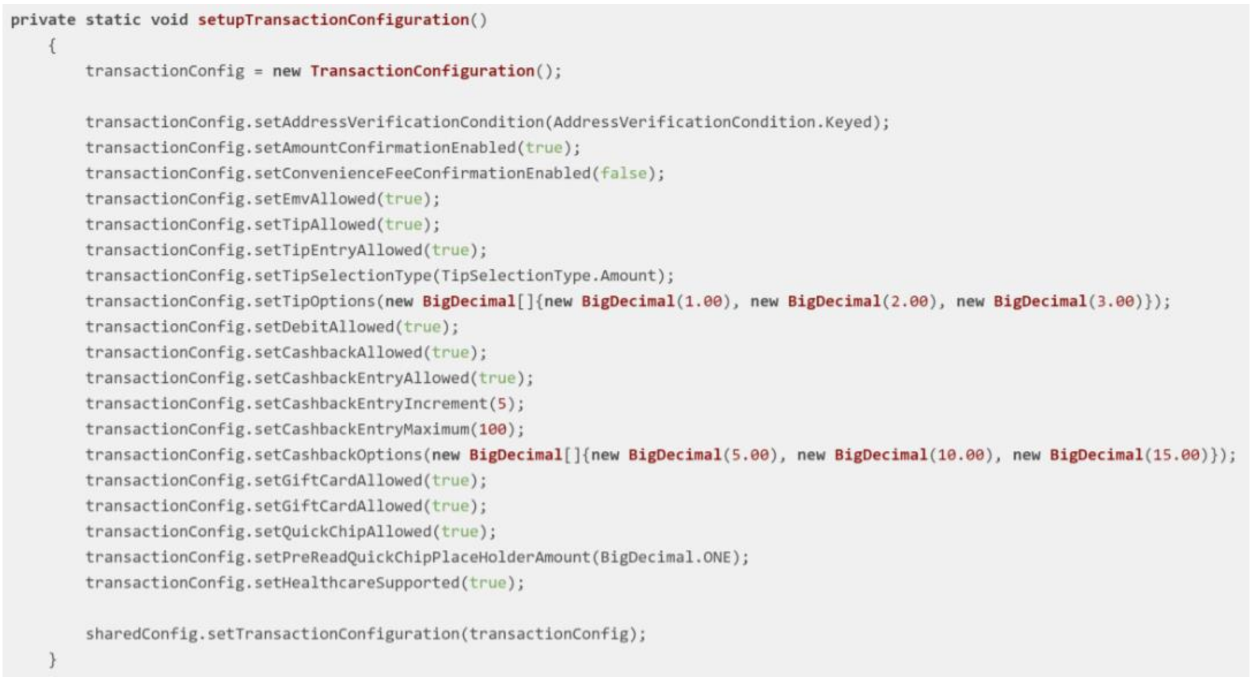
setUpDeviceConfiguration

- The setUpDeviceConfiguration method is used to configure the required settings for the target device and connection types and set identifiable properties for the terminal.
- bluetoothConfiguration: set the BT identifier as displayed on the Android Tablet.
- TcpIpConfiguration: set the IP address and Port number on the POI device.

```
private static void setupDeviceConfiguration()
    {
        // Device configuration
        deviceConfig = new DeviceConfiguration();

        deviceConfig.setContactlessAllowed(true);
        deviceConfig.setDeviceType(DeviceType.IngenicoUppBluetooth);
        deviceConfig.setKeyedEntryAllowed(true);
        deviceConfig.setTerminalId("1234");
        deviceConfig.setTerminalType(TerminalType.Mobile);

        // Bluetooth configuration
        BluetoothConfiguration bluetoothConfiguration = new BluetoothConfiguration();
        bluetoothConfiguration.setIdentifier("LK2500_12345678");
        deviceConfig.setBluetoothConfiguration(bluetoothConfiguration);

        // TCP/IP configuration
        TcpIpConfiguration tcpIpConfiguration = new TcpIpConfiguration();
        tcpIpConfiguration.setIpAddress("10.0.1.16");
        tcpIpConfiguration.setPort(12000);
        deviceConfig.setTcpIpConfiguration(tcpIpConfiguration);

        sharedConfig.setDeviceConfiguration(deviceConfig);
    }
```

setUpTransactionConfiguration

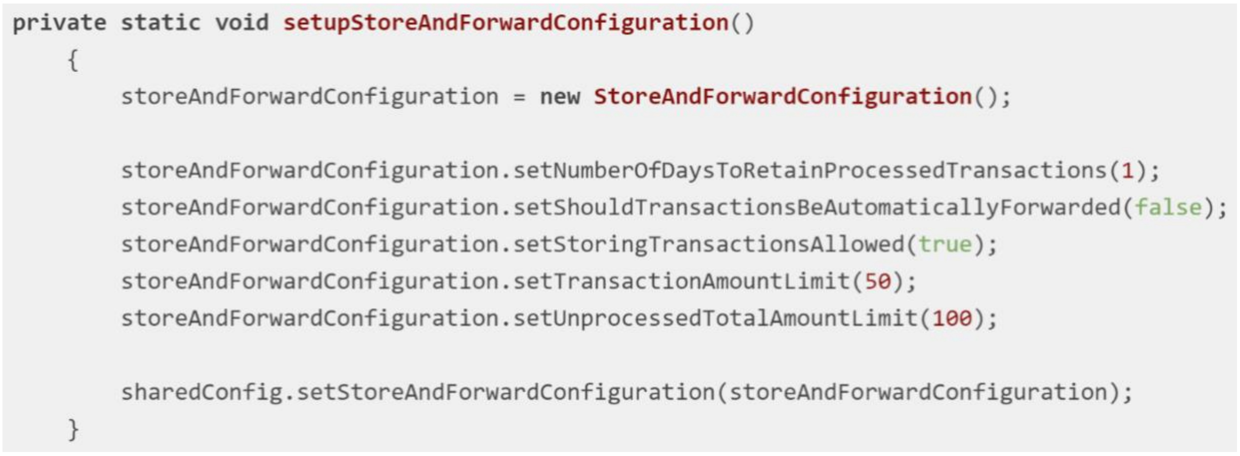
The setUpTransactionConfiguration method is used to set all the transactionConfiguration related properties that change functionality of transaction flows.



```
private static void setupTransactionConfiguration()
    {
        transactionConfig = new TransactionConfiguration();

        transactionConfig.setAddressVerificationCondition(AddressVerificationCondition.Keyed);
        transactionConfig.setAmountConfirmationEnabled(true);
        transactionConfig.setConvenienceFeeConfirmationEnabled(false);
        transactionConfig.setEmvAllowed(true);
        transactionConfig.setTipAllowed(true);
        transactionConfig.setTipEntryAllowed(true);
        transactionConfig.setTipSelectionType(TipSelectionType.Amount);
        transactionConfig.setTipOptions(new BigDecimal[]{new BigDecimal(1.00), new BigDecimal(2.00), new BigDecimal(3.00)});
        transactionConfig.setDebitAllowed(true);
        transactionConfig.setCashbackAllowed(true);
        transactionConfig.setCashbackEntryAllowed(true);
        transactionConfig.setCashbackEntryIncrement(5);
        transactionConfig.setCashbackEntryMaximum(100);
        transactionConfig.setCashbackOptions(new BigDecimal[]{new BigDecimal(5.00), new BigDecimal(10.00), new BigDecimal(15.00)});
        transactionConfig.setGiftCardAllowed(true);
        transactionConfig.setGiftCardAllowed(true);
        transactionConfig.setQuickChipAllowed(true);
        transactionConfig.setPreReadQuickChipPlaceHolderAmount(BigDecimal.ONE);
        transactionConfig.setHealthcareSupported(true);

        sharedConfig.setTransactionConfiguration(transactionConfig);
    }
```

setUpStoreAndForwardConfiguration

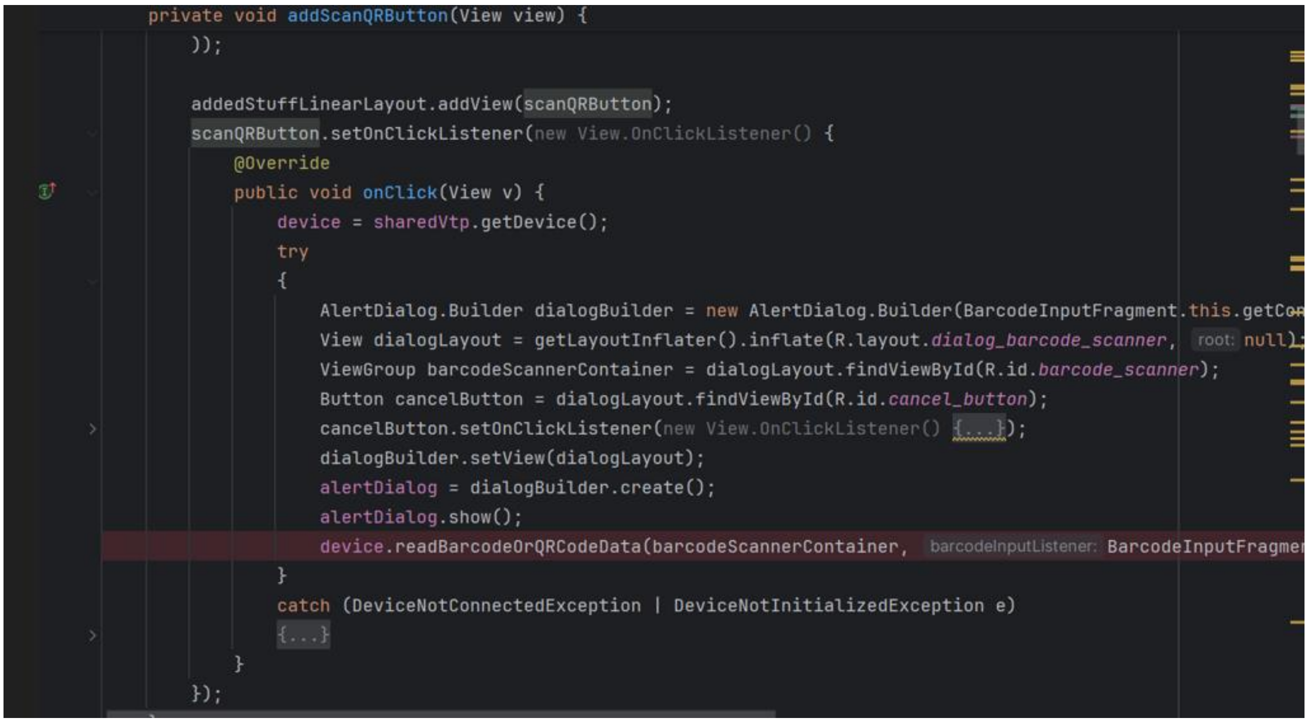
The setUpStoreAndForwardConfiguration method is used to set all the store and forward related settings.



```
private static void setupStoreAndForwardConfiguration()
    {
        storeAndForwardConfiguration = new StoreAndForwardConfiguration();

        storeAndForwardConfiguration.setNumberOfDaysToRetainProcessedTransactions(1);
        storeAndForwardConfiguration.setShouldTransactionsBeAutomaticallyForwarded(false);
        storeAndForwardConfiguration.setStoringTransactionsAllowed(true);
        storeAndForwardConfiguration.setTransactionAmountLimit(50);
        storeAndForwardConfiguration.setUnprocessedTotalAmountLimit(100);

        sharedConfig.setStoreAndForwardConfiguration(storeAndForwardConfiguration);
    }
```

QR Code and Barcode Scanning Functionality

The addScanQRButton method is used to enable the QR Code and Barcode scanner.



```
private void addScanQRButton(View view) {
    ));

    addedStuffLinearLayout.addView(scanQRButton);
    scanQRButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            device = sharedVtp.getDevice();
            try
            {
                AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(BarcodeInputFragment.this.getCon
                View dialogLayout = getLayoutInflater().inflate(R.layout.dialog_barcode_scanner,  root: null);
                ViewGroup barcodeScannerContainer = dialogLayout.findViewById(R.id.barcode_scanner);
                Button cancelButton = dialogLayout.findViewById(R.id.cancel_button);
                cancelButton.setOnClickListener(new View.OnClickListener() {...});
                dialogBuilder.setView(dialogLayout);
                alertDialog = dialogBuilder.create();
                alertDialog.show();
                device.readBarcodeOrQRCodeData(barcodeScannerContainer,  barcodeInputListener: BarcodeInputFragmen
            }
            catch (DeviceNotConnectedException | DeviceNotInitializedException e)
            {...}
        }
    });
```