

**POSitive Integration Manual**

Document Version: 0.0.2

POSitive Application Version: 0.0.2

last updated 21/08/2019

by

EFT Solutions Ltd

www.eft-solutions.co.uk

Table of Contents

[1. Confidentiality Statement 3](#_Toc14730622)

[2. Change History 3](#_Toc14730623)

[3. Glossary 4](#_Toc14730624)

[4. Introduction 5](#_Toc14730625)

[5. A920 3rd Party Integration 6](#_Toc14730626)

[1. Prerequisites 6](#_Toc14730627)

[2. POSitiveLauncher – Sample Application 6](#_Toc14730628)

[3. Terminal Profile 6](#_Toc14730629)

[4. Integration Rules 7](#_Toc14730630)

[5. Transaction Initiation 7](#_Toc14730631)

[6. Transaction Results 8](#_Toc14730632)

# Confidentiality Statement

© 2019, EFT Solutions Ltd

[warren@eft-solutions.co.uk](mailto:warren@eft-solutions.co.uk)

ALL RIGHTS RESERVED. This document contains material protected under International and Federal Copyright Laws and Treaties. Any unauthorised reprint or use of this material is prohibited. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without express written permission from EFT Solutions Ltd.

# Change History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Who | Ver | Change Description |
| 19/7/19 | Philip Clarkson | 0.0.1 | Initial Draft |
| 21/08/19 | Philip Clarkson | 0.0.2 | Updated to use POSIntegrate (new interface) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Glossary

|  |  |
| --- | --- |
| Term | Definition |
| UTI | Unique Transaction Identifier (used for transaction lookup) |
| MSR | Mag Stripe Transaction (Card Swiped) |
| CTLS | Contactless Transaction (Card Tapped) |
| EMV | Insert card transaction |
| Reversal | Used for a full cancellation of a previous transaction |
| Refund | Used to credit a cardholders account (Not linked to previous sale) |
| Sale | A purchase transaction that debits the cardholders account |
| PreAuth | A transaction to reserve funds on an account |
| Completion | A transaction to complete a previous pre-auth and transfer the funds permanently |

# Introduction

This document is intended to be read by application integrators, who want to integrate card payments into their Android-based Point of Sale application via the POSitiveLib library.

POSitiveLib provides an easy-to-use API interface to initiate card transactions, in a java library.

# A920 3rd Party Integration

This document provides the instructions for the POSitiveLauncher demo application required for 3rd party developers to write applications that integrate with POSitive on the Pax A920 payment terminal.

## Prerequisites

1. A Pax A920 payment terminal.
2. A USB cable (USB-A to USB micro-B).
3. The terminal must be installed with at least version 1.00.00 of the POSitive payment application, ideally configured in demo mode to assist development.
4. The terminal must be put into debug mode using the Pax website (PAX developers should know how to do this, so instructions have not been added, see Pax for more details)

## POSitiveLauncher – Sample Application

Unzip and import the POSitiveLauncher package into Android Studio.

1. The application provides a very simple program that will allow you to see how a transaction and a reversal should be performed. Test buttons are provided on the main activity (see MainActivity.java)
2. Comments have been added in the code to explain how it works, but it should be quite self-explanatory
3. The code relies on a library that is included called POSitivelib-X.X-release.aar. Look at the manifest and the build.gradle files to see how it is integrated.
4. The IPC comms rely on a BroadcastReceiver being used to send/receive the messages. Look at the manifest to see how this should be declared. The example implementation is in TestLaunchReceiver.java
5. It is the responsibility of the calling app to return itself to the foreground once the transaction is completed (The sample app also does this in the POSitiveLauncReceiver by calling startActivity() once the result has been received)
6. EFT Solutions intend to update the POSitiveLib with more functionality as required, so more details of the transaction will be added as requested. Please email specific requests/comments to philip@eft-solutions.co.uk

In Summary: you should be able to declare a receiver and use the included library to do IPC comms to and from the POSitive app (Using this app as a working example)

## Terminal Profile

There is a config file on the terminal called profile.xml. This file determines which apps are on display from the main menu.

There is an example of the proile.xml file containing POSitiveLauncher in the root directory of the test app.

The file needs to contain the name you want to display on the menu and the package name, so that the main service can identify your application.

E.g.

<**menuItem**>  
 <**packageName**>eft.com.positivelauncher</**packageName**>  
 <**displayName**>Test Launcher</**displayName**>  
</**menuItem**>

You can update the file and install it to display your app using the following adb.exe command. In a live environment this configuration will be pushed down from the TMS.

adb.exe push profile.xml /data/data/eft.com.positivesvc/files

## ****Integration Rules****

1. The apps must not bring down the network stack. Both the payment application and the paxstore app require internet connectivity to upload transactions in the background.

2. New applications must not install themselves as a launcher process

## Transaction Initiation

To initiate a transaction, call the API functions on the PosIntegrate class. The methods take additional configuration, passed in as a HashMap to the function.

The HashMap is populated with additional values identified by the enum CONFIG\_TYPE on PosIntegrate.java

Declarations:

* PositiveError **executeTransaction**(Context context, TRANSACTION\_TYPE transType, HashMap<CONFIG\_TYPE, String> args)
* PositiveError **cancelTransaction**(Context context)
* PositiveError **executeReversal**(Context context, HashMap<CONFIG\_TYPE, String> args)

For Example.

This will add an amount to the hashmap of 100 (minor units)

HashMap<CONFIG\_TYPE, String> args = **new** HashMap<CONFIG\_TYPE, String>();  
args.**put**(***CT\_AMOUNT***, **"100"**);

* SALE:
  + PosIntegrate . executeTransaction (this, TRANSACTION\_TYPE\_SALE, args);
* REFUND:
  + PosIntegrate . executeTransaction (this, 100, TRANSACTION\_TYPE\_REFUND, args);
* REVERSAL:
  + PosIntegrate .executeReversal(this, args);
* CANCEL:
  + PosIntegrate.cancelTransaction(this);
* QUERY:
  + PosIntegrate.queryTransaction(this);

Reverse transactions based on their UTI, or just reverse the last transaction by passing in a receipt number of 0.

Pass additional arguments to disable printing, and keep the reversal silent (in the background)

**Purchase With Cashback:** The sale transaction will offer cashback where the terminals configuration indicates that cashback is supported for the card type presented. If you want to enable\disable this feature use paxstore to update your terminals settings. You can also pass in a cashback or gratuity amount and the app will use it if allowed by the card.

## Transaction Results

Different results can come back from the POSitive app based on the transaction result.

The following method is used to unpack the results into an object.

* PositiveTransResult **unpackResult**(Context context, Intent intent)

The results on the object are split into four separate lists, with a flag that can be checked on the object to indicate if the list is present.

E.g.

For a successful CTLS transaction, lists 1-3 would be returned.

For a successful MSR transaction, lists 1 and 2 would be returned.

For a critical failure resulting from a crash or a programming error then list 4 would be returned.

List One is returned when the **transResponse** boolean = true

List Two is returned when the **transDetails** boolean = true

List Three is returned when the **cardType** String = “EMV” or “CTLS”

List Four is returned when there is a serious failure and **transResponse** = false;

The broadcast receiver has code to demonstrate extracting the results for the calling app to use.

1. **List One - Standard Response Details (transResponse = true)**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Date Type** | **Descriptio**n |
| UTI | GUID | Unique Transaction Identifier (e.g. 5594801e-a3e5-da11-8b4600065b3e6c8d) |
| amountTrans | long | Amount of the transaction (minor units, 100 = £1.00) |
| amountGratuity | long | The tip amount (minor units, 100 = £1.00) |
| amountCashback | long | The cashback amount (minor units, 100 = £1.00) |
| transApproved | Boolean | Transaction result |
| transCancelled | Boolean | Set when the user manually cancels the transaction. |
| cvmSigRequired | Boolean | Indicator to let called know if signature is required |
| cvmPinVerified | Boolean | Indicator to say if the pin was verified |
| transCurrencyCode | String | 3 character currency code (GBP/EUR etc) |
| terminalId | String | The terminal soft ID (e.g. 12345678) |
| merchantId | String | The merchant ID (e.g. 123456789012345) |
| softwareVersion | String | The software version (e.g. 1.00.00) |

1. **List Two (transDetails = true)**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| **receiptNumber** | **int** | **The number of the receipt** |
| **retrievalReferenceNumber** | **String** | **The retrieval reference number (matches the RRN for the protocol being used)** |
| **responseCode** | **String** | **The response code from the acquirer (e.g. 00 = Approved)** |
| **stan** | **String** | **The stan used in the protocol messages** |
| **authorisationCode** | **String** | **The auth code returned from the acquirer (e.g. 123ABC)** |
| **merchantTokenId** | **String** | **Token from the gateway (if sent)** |
| **cardPan** | **String** | **The masked pan of the card (e.g. 545454\*\*\*\*\*\*5454)** |
| **cardExpiryDate** | **String** | **The expiry date of the card used – YYMM** |
| **cardStartDate** | **String** | **The start date of the card used (if available) – YYMM** |
| **cardScheme** | **String** | **The name of the scheme used (e.g. Visa\Mastercard)** |
| **cardPanSequenceNumber** | **String** | **The pan sequence number (e.g. 001)** |
| **cardType** | **String** | **The capture method of the card (EMV/MSR/Contactless/Manual)** |

1. **List Three - EMV Details**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **EMV tag data source** | **Description** |
| emvAid | String | 9f06 | The application identifier (e.g. A0000000031010) |
| emvTsi | String | 9b | Transaction Status Information (e.g. E800) |
| emvCardholderName | String | 5f20 | The cardholder name (if available, e.g. P CLARKSON) |
| emvCryptogram | ByteArray | 9f26 | The cryptogram used (e.g. 8754EA78EB65AB65) |
| emvCryptogramType | String | Derived from 9f27 | The type of the cryptogram (e.g. AAC/TC/ARQC) |

4. **List Four – Error Details**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| TransResponse | Boolean | Always false as there has been an error |
| Approved | Boolean | Always false as there has been an error |
| Error | Enum | See the PositiveErrors enum type in the library |
| ErrorText | String | Additional error information |

## Transaction Reports

To initiate a report, call the following API function on the PosIntegrate class.

The HashMap is populated with the report type

Declarations:

* PositiveError **executeReport**(Context context, TRANSACTION\_TYPE transType, HashMap<CONFIG\_TYPE, String> args)
* PositiveError **unpackReport**(Context context, Intent intent)

For Example.

This will add the report type to the hashmap

HashMap<CONFIG\_TYPE, String> args = **new** HashMap<CONFIG\_TYPE, String>();  
args.**put**(***CT\_XREPORT***, **"TRUE"**);

* RUN REPORT:
  + PosIntegrate . executeReport (this, args);
* UNPACK REPORT RESULTS (from receiver)
  + PosIntegrate . unpackReport (content, intent);

The report should be printed by the terminal and the results returned to the broadcast receiver. The unpackreport method can be used to extract the results into an object.