



Android Developer Integration Guide
Version 1.8

PayGuardian Android Developer Integration Guide

Version 1.8 Released February 15, 2017

Copyright © 2011-2017, BridgePay Network Solutions, Inc. All rights reserved.

The information contained herein is the confidential and proprietary property of BridgePay Network Solutions, Inc. and may not be used, distributed, modified, disclosed, or reproduced without express written permission.

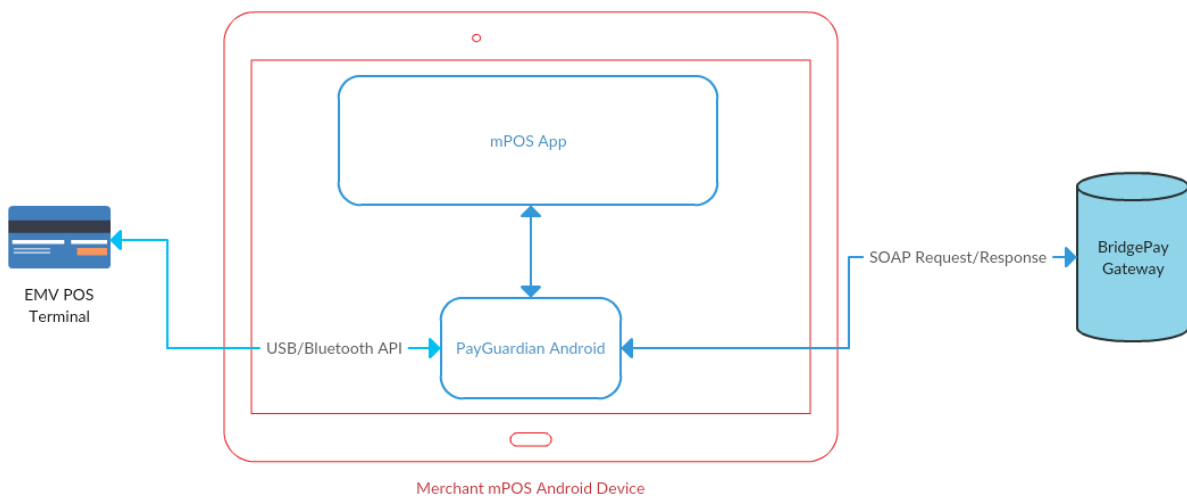
Table of Contents

Introduction.....	3
1. PayGuardian Setup	4
1.1. Requirements	4
1.2 Android Application Setup Guide.....	5
1.2.1 Minimum Requirements	5
1.2.2 Setup	5
1.2.3 Change Processing Mode	5
1.3 Android Library (SDK) Setup Guide	8
1.3.1 Minimum Requirements	8
1.3.2 Setup	8
1.4 Android Device Configuration	9
2. Integration Process	10
2.1. Getting Started	10
2.2. Integration Options	10
3. PayGuardian Integration	11
3.1. Integration Methods	11
3.1.1. App-to-App Communication Method	11
3.1.2. Custom URI Method.....	12
3.1.3. PayGuardian Library (SDK) Integration	14
3.1.4. Program Input Object	16
3.1.5. Program Output Object	18
3.1.6. PayGuardian Android XML Data Format	19
3.1.6.1. Examples	19
3.2. EMV Transaction Support.....	31
3.2.1. EMV Implementation Details	31
3.2.2. EMV Purchase Transaction Flow	32
A. Appendix.....	35
A.1. Response Values.....	35
A.1.1. Result Codes	35
A.1.2. AVS Response Codes.....	35
A.1.3. CV Response Codes.....	36
A.2. Errors.....	37
A.2.1. App to App Communication Errors.....	37
A.3. Test Cards and Data	38
A.4. Terms of Use Agreement.....	41

Introduction

Mobile Point-of-Sales (mPOS) systems that process sensitive payment information are required to certify their payment applications to the Payment Application Data Security Standard (PA-DSS). The addition of EMV certification and continued need for both encryption and tokenization has become a concern for both merchants and integrators. Instituting and maintaining these standards requires significant financial and employee resources in order to adhere to the Payments Card Industry Data Security Standards (PCI DSS) compliance requirements. Subsequently, any changes made in the mPOS system, may require a partial or full recertification, which increases the cost and time to market. BridgePay has engaged these issues through our product line, the Pay Guardian suite, to better serve the needs of our integrators and merchants.

PayGuardian Android is a light weight, highly secure Android application that integrates seamlessly into mPOS applications. PayGuardian Android facilitates the transaction process by handling the collection and transmission of sensitive payment information as an out of scope PA-DSS solution, thereby offloading the certification responsibility from merchants and integrators. PayGuardian Android is EMV enabled, handles point to point encryption, and tokenizes all transactions to ensure transaction processing is seamless and secure.



- ✓ The mPOS application collects transaction data such as the dollar amount, invoice number, tender type, transaction type, merchant account information, etc...
- ✓ The mPOS application constructs the transaction request in an XML format upon the selection of the Process button. (See [3.1.3. Sample examples](#))
- ✓ PayGuardian obtains the card information via the claiming of the mPOS device using the Device USB Host connection (re: Ingenico RBA SDK) and validates the transaction request object.
- ✓ The mPOS device prompts to swipe or insert the card and transmits the card information to the PayGuardian application, which captures the card data as a swiped or EMV transaction respectively.
- ✓ PayGuardian constructs the payload request and transmits the transaction request to the Payment Gateway.
- ✓ PayGuardian transmits the transaction response returned from the Payment Gateway back to the mPOS application.
- ✓ The mPOS application implementation responds accordingly to the transaction response.

1. PayGuardian Setup

1.1. Requirements

Operating Systems

PayGuardian has a PA-DSS certification for the following operating systems:

- ✓ Android SDK version 17 (Jellybean 4.2) and above
- ✓ Android device should have the USB host feature, Audio Jack, or Bluetooth connection
- ✓ Ingenico card reader device - RBA version 15.0.4
- ✓ BBPOS device Chipper 2 or WisePad 2 - terminal software
- ✓ IDTech device Unipay III - terminal software NEO v1.01.023

1.2 Android Application Setup Guide

1.2.1 Minimum Requirements

- Android SDK version 17 (Jellybean 4.2) and above
- Android device should have the USB host feature, Audio Jack, or Bluetooth connection
- Ingenico card reader device - RBA version 15.0.4
- BBPOS device Chipper 2 or WisePad 2 - terminal software
- IDTech device Unipay III - terminal software NEO v1.01.023

1.2.2 Setup

Install the PayGuardian application into the device.

App Store: TBD

Manual Setup:

Install the application onto the device.

Step1: Copy the PayGuardian Apk file to the Android device

Step2: Go to 'Settings', scroll down to 'Security', and select 'Unknown sources'. Selecting this option allows the app installation to occur outside of the Google Play store. Some devices permit the selection of the option to be warned before installing harmful applications. This feature can be enabled by selecting the 'Verify applications' option in the Security settings.

Step3: Click the Apk file to install the application.

 PayGuardian Android supports Portrait and Landscape orientation.

1.2.3 Change Processing Mode

The PayGuardian Android application has both a Test and Production mode that can be switched as needed. The Test mode setting should always be used for integration and test purposes.

PayGuardian Android offers two methods of switching the gateway URL between Production and Test mode, either through an integration or via the PayGuardian Android application.

PayGuardian Android Application - Integrated Processing Mode Switch

PayGuardian Android allows integrators to remotely change the gateway URL between the production and test environments. An integer is sent via an intent where

0 = TEST and 1 = PRODUCTION.

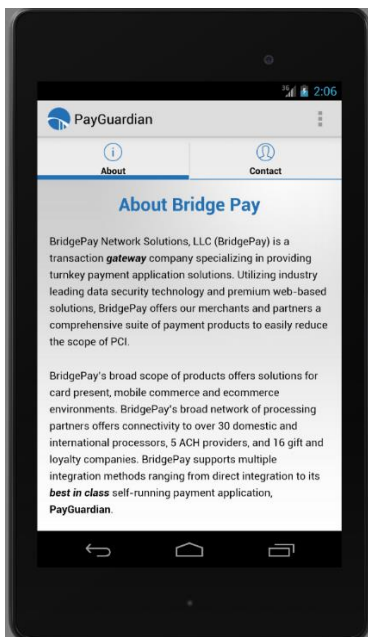
To accomplish this, use the following example:

```
Intent setModelIntent = new Intent();  
  
setModelIntent.putExtra("MODE", 0);  
  
setModelIntent.setComponent(new ComponentName("com.bpn.payguardian",  
"com.bpn.payguardian.service.ChangeModeService"));  
  
startService(setModelIntent);
```

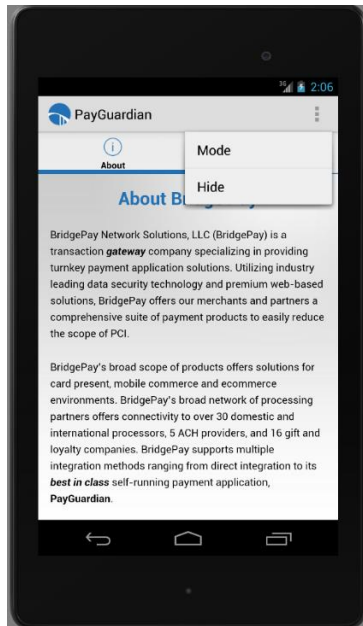
This will execute the remote IntentService and set the gateway URL accordingly without interrupting the mPOS application.

PayGuardian Android Application - Manual Processing Mode Switch

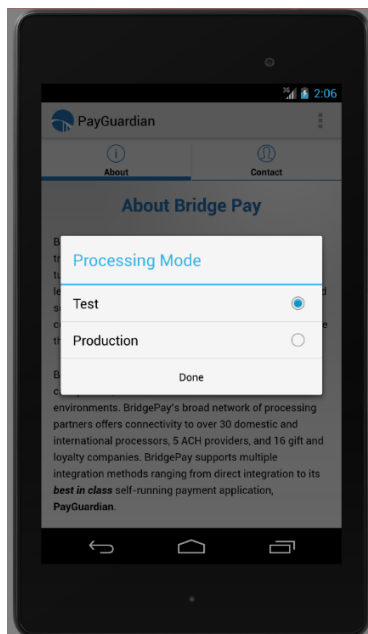
Step1: On the main PayGuardian App screen, press the Menu button located in the upper right hand corner of the screen (see mainScreen.PNG).



Step2: Select the Mode item (Menu.PNG)



1. **Step3:** The current processing mode will be highlighted in the list display. Select the new processing mode option and press the Done button (see setMode.PNG) to save the changes. If no changes were made, close the menu. The current selected processing mode will remain unchanged.




1.3 Android Library (SDK) Setup Guide

1.3.1 Minimum Requirements

- Android SDK version 17 (Jellybean 4.2) and above
- Android device should have the USB host feature, Audio Jack, or Bluetooth connection
- Ingenico card reader device - RBA version 15.0.4
- BBPOS device Chipper 2 or WisePad 2 - terminal software
- IDTech device Unipay III - terminal software NEO v1.01.023

1.3.2 Setup

 The instructions below are based on the assumption that Android Studio 1.3.2 or later will be used in the development environment. The directions will differ for alternate development environments and can be obtained by contacting Developer Support.

Step1: Copy the payguardianandroid-release.aar file to your project's /libs folder.

Step2: Select "New Module" option under the File menu.

Step3: Select "Import .JAR/.AAR Package" and click next.

Step4: Select the path to the payguardianandroid-release.aar file and click Finish.

Step5: Select "Project Settings" under the File menu.

Step6: Under "Modules", in the left menu, select "app".

Step7: Click the "Dependencies" tab.

Step8: Click the green "+" symbol in the upper right corner.

Step9: Select "Module Dependency".

Step10: Select the new PayGuardian module from the list display.

1.4 Android Device Configuration

- Android device and card reader should be connected through either USB Host connectivity, Audio Jack, or Bluetooth pairing options.

1.4.1 Ingenico

USB Supported devices: **iPP350 and iPP320**

The Android device and card reader should be connected through the USB Host connection via the OTG (On the Go) cable.

Bluetooth Supported devices: **iCMP, and iSMP**

The Android device and card reader should be paired through the Bluetooth connection.

1.4.2 BBPOS

Audio Jack Supported devices: **Chipper 2**

The Android device and card reader should be connected through the audio jack.

Bluetooth Supported devices: **WisePad 2**

The Android device and card reader should be paired through the Bluetooth connection.

1.4.2 IDTech

Audio Jack Supported devices: **Unipay III**

The Android device and card reader should be connected through the audio jack.

2. Integration Process

2.1. Getting Started

The PayGuardian application should be installed in the Android device prior to the start of an integration.

Contact: Developer.Support@bridgepaynetwork.com to receive the developer application build of PayGuardian Android. The developer application build points to the PayGuardian UAT environment for integration and certification testing.

2.2. Integration Options

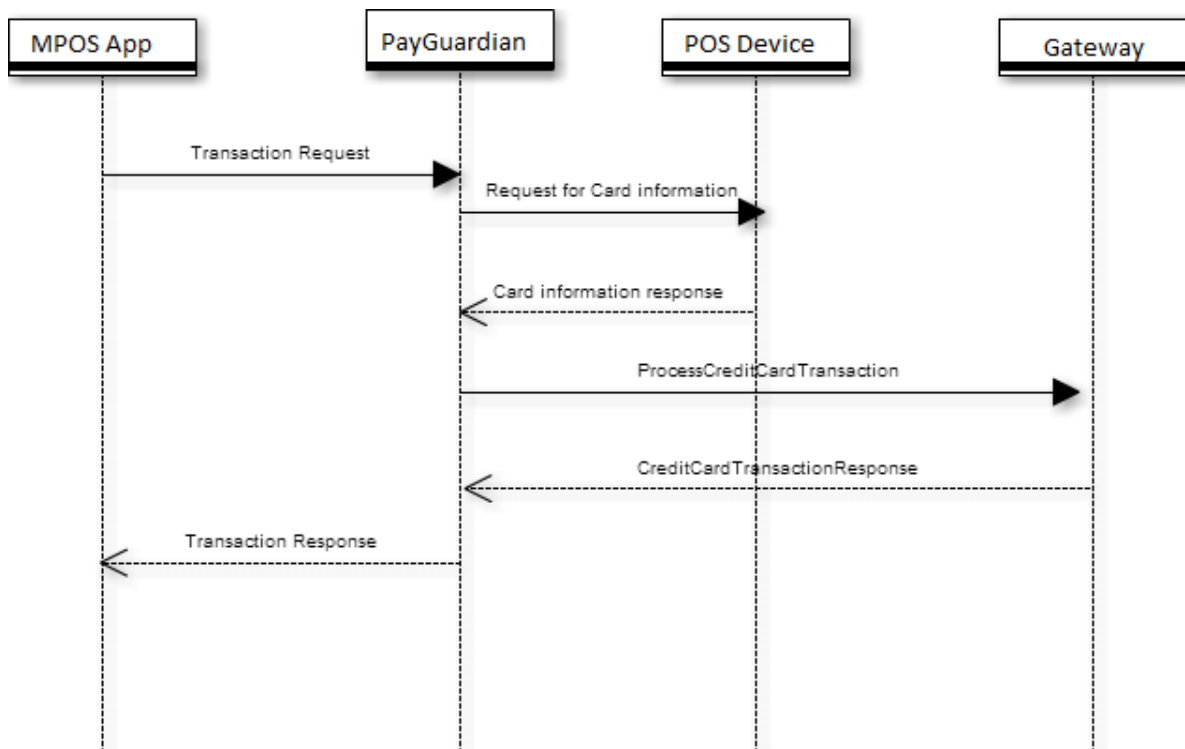
PayGuardian Android currently offers several integration options: **App-to-App communication, Custom URI, or direct to Library (SDK)**. Please note that the enumerations, properties and methods contained within this document are only for a PayGuardian Android integration.

- ① Remember that the application integrating to PayGuardian Android should never touch, collect, transmit, or store the account holder's actual card information.
- ① The integrating application must always pass a unique identifier for each mPOS user or cashier.

2.3. Transaction Flow Summary

The following list summarizes the steps involved in processing a transaction:

1. The mPOS system collects order information and determines that the customer will pay with one of the supported tender types.
2. The mPOS system invokes the PayGuardian app method.
3. The PayGuardian payment screen loads and prompts the user to swipe or insert card.
4. After collecting all transaction information, PayGuardian transmits the data to the server.
5. After receiving a response from the host, the payment screen returns the processing results back to the mPOS system.
6. The mPOS system analyses the response data.



3. PayGuardian Integration

3.1. Integration Methods

By using App to App communication or Custom URI integration methods, PayGuardian Android ensures that your application remains completely out of scope of the Payment Application Data Security Standard (PA-DSS) certification requirement. When a transaction is initialized, the PayGuardian Android application will place itself in the foreground while the request is being processed. Conversely, when the response is returned, it is transmitted back to the requesting application which is then placed into the foreground.

3.1.1. App-to-App Communication Method

The PayGuardian Android application is accessed by starting an activity with an intent which encapsulates a request payload.

NOTE* Where paymentRequestXml is a String representation of the Program Input Object (currently) in section 3.1.1

```

public void processTransaction(String paymentRequestXml)
{
    final int PAYGUARDIAN_REQUEST_CODE = 102;
    Intent sendIntent = new Intent(android.content.Intent.ACTION_SEND);
    sendIntent.setType("text/plain");
    sendIntent.putExtra(Intent.EXTRA_TEXT, paymentInputXml);

    PackageManager pm = getApplicationContext().getPackageManager();
    List<ResolveInfo> activityList = pm.queryIntentActivities(sendIntent, 0);

    for (final ResolveInfo app : activityList)
    {
        if ((app.activityInfo.name).contains("ProcessSecurelyActivity"))
        {
            final ActivityInfo activity = app.activityInfo;
            final ComponentName name = new
ComponentName(activity.applicationInfo.packageName, activity.name);

            sendIntent.setComponent(name);

            startActivityForResult(sendIntent, PAYGUARDIAN_REQUEST_CODE);

            break;
        }
    }
}

```

3.1.2. Custom URI Method

The custom URI integration method can be used from both native android apps and non-native apps (such as a web application). There is an additional required field for the Custom URI integration method named “callbackUri”. This is the URI (associated with your application) where PayGuardian will post the payment response.

If the request originated from a native app intent, the response will be returned via an Intent with the data field containing the callbackUri, and with the response XML serialized in the PaymentResponse element of the query string.

If the request originated from a non-native app, the response will be returned to a web browser via a BrowserIntent using the callbackUri provided. The request is a String representation of the Program Input Object XML and must have no new lines, line feeds, or whitespace between elements. The request must be URL encoded.

The custom URI must match the format:

bpn://bridgepaynetwork.com/transact?PaymentRequest=ProgramInputObjectXML

A ProgramInputObjectXML example is:

```
%3CPaymentRequest%3E%3CTenderType%3ECREDIT%3C%2FTenderType%3E%3CTransType%3ESALE%3C%2FTransType%3E%3CAmount%3E5.00%3C%2FAmount%3E%3CUsername%3EMerchantUser%3C%2FUsername%3E%3CPassword%3EMerchantPass%3C%2FPassword%3E%3CZip%2F%3E%3CStreet%2F%3E%3COrigRefNum%2F%3E%3CCountry%2F%3E%3CPhone%2F%3E%3CEmail%2F%3E%3CMerchantCode%3EMerchantCode%3C%2FMerchantCode%3E%3CMerchantAccountCode%3EMerchantAccountCode%3C%2FMerchantAccountCode%3E%3CInvNum%3E1234%3C%2FInvNum%3E%3CCallbackUri%3Eyour%3A%2F%2Fcustom%2Furi%3C%2FCallbackUri%3E%3CReferenceID%3E555%3C%2FReferenceID%3E%3CTerminalType%3Ekeyed%3C%2FTerminalType%3E%3C%2FPaymentRequest%3E
```

This string, again, is a URL encoded representation of the Program Input Object XML detailed in section #####. An unencoded version of this string, for reference, looks like:

```
<PaymentRequest>
  <TenderType>CREDIT</TenderType>
  <TransType>SALE</TransType>
  <Amount>5.00</Amount>
  <Username>MerchantUser</Username>
  <Password>MerchantPass</Password>
  <Zip/>
  <Street/>
  <OrigRefNum/>
  <Country/>
  <Phone/>
  <Email/>
  <MerchantCode>MerchantCode</MerchantCode>
  <MerchantAccountCode>MerchantAccountCode</MerchantAccountCode>
  <InvNum>1234</InvNum>
  <CallbackUri>your://custom/uri</CallbackUri>
  <ReferenceID>555</ReferenceID>
  <TerminalType>keyed</TerminalType>
</PaymentRequest>
```

Example of sending custom URI via native-app Intent:

```
String encodedProgramInputObject =
"\bnpn://bridgepaynetwork.com/transact?PaymentRequest=%3CPaymentRequest%3E%3CTe
nderType%3ECREDIT%3C%2FTenderType%3E%3CTransType%
3ESALE%3C%2FTransType%3E%3CAmount%3E5.00%3C%2FAmount%3E%3CUsername%3EM
erchantUser%3C%2FUsername%3E%3CPassword%3EMerchantPass%3C%2FPassword%3E%3
CZip%2F%3E%3CStreet%2F%3E%3COrigRefNum%2F%3E%3CCountry%2F%3E%3CPhone%2F
%3E%3CEmail%2F%3E%3CMerchantCode%3EMerchantCode%3C%2FMerchantCode%3E%3C
MerchantAccountCode%3EMerchantAccountCode%3C%2FMerchantAccountCode%3E%3CInv
Num%3E1234%3C%2FInvNum%3E%3CCallbackUri%3Eyour%3A%2F%2Fcustom%2Furi%3C%2
FCallbackUri%3E%3CReferenceID%3E555%3C%2FReferenceID%3E%3CTerminalType%3Ekeye
d%3C%2FTerminalType%3E%3C%2FPaymentRequest%3E\"";

Intent intent = new Intent(android.content.Intent.ACTION_SEND);
intent.setData(Uri.parse(encodedProgramInputObject));
```

```

try
{
    startActivityResult(intent, PAYGUARDIAN_REQUEST_CODE);
}
catch (Exception e)
{
    //Activity not found
}

```

3.1.3. PayGuardian Library (SDK) Integration

The PayGuardian Android SDK gives you direct access to the PayGuardian API directly within your application without the need to call an activity of a separate application. The SDK provides a simple convenience class aptly named PayGuardian, that makes a simple integration easy.

The PayGuardian class constructor is:

```
public PayGuardian(Context context, PayGuardianCallback callback)
```

The constructor has 2 arguments of the types: android.content.Context (the calling application context) and PayGuardianCallback. PayGuardianCallback is an interface that is implemented by the calling application to receive notifications on status, results, or errors from the PayGuardian SDK.

The PayGuardianCallback has 2 methods:

```
void statusUpdated(PayGuardianStatus status);
```

```
void transactionResponse(PaymentResponse response);
```

PayGuardianStatus is an enumeration of different status events, so that the calling application can update its UI or other items, as the status of a transaction changes.

PaymentResponse is the Program Output Object detailed in section 3.1.5

The PayGuardian class contains one method to initialize a transaction:

```
public boolean processTransaction(PaymentRequest request, boolean testMode)
```

The “processTransaction” method is non-blocking and will return a Boolean value indicating a success, or failure, of initializing a transaction. This method will only return false in the event that there is an unrecoverable exception (NullPointerException) during initialization. There are two arguments to the method. First, the PaymentRequest object (detailed in section 3.1.4 Program Input Object). Second, the test mode flag that will tell PayGuardian whether or not this is a live transaction.

As a transaction navigates through various steps (communicating with the terminal, configuring terminal, communicating with the gateway, etc.), the “statusUpdated” callback method will be called with the status change information.

When a transaction completes (either success or error), the transactionResponse callback method will be called with the PaymentResponse object. In the case of an error, the error code and message will be available in the response object as detailed in section 3.1.5

SDK Integration Code Example

```
void testTransaction()
{
    PaymentRequest testRequest = new PaymentRequest();
    testRequest.setAmount("5.00");
    testRequest.setMerchantAccountCode("12345");
    testRequest.setMerchantCode("12345");
    testRequest.setUsername("merchantUsername");
    testRequest.setPassword("merchantPassword");
    testRequest.setInvNum("54321");
    testRequest.setTransType(PayGuardian.TransactionTypes.SALE.toString());
    testRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.toString());
    testRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB); //Ingenico USB
    Terminal

    PayGuardian payGuardian = new
    com.bpn.payguardian.android.PayGuardian(getApplicationContext(), this);

    boolean started = payGuardian.processTransaction(paymentRequest, true);

    if (started == true)
    {
        //successfully started trx
    }
    else
    {
        //error condition
    }
}

public void statusUpdated(PayGuardianStatus status)
{
    //handle status update event
}

public void transactionResponse(PaymentResponse response)
{
    //handle transaction response event
}
```

3.1.4. Program Input Object

The following table contains descriptions for the properties of the extension input object. The properties are optional unless otherwise indicated.

Property	Description	Data Type / Min - Max Values
TenderType	Required. Valid values are: CREDIT DEBIT GIFT	String; Min Value = 4; Max Value = 25
TransType	Required. Valid values are: SALE : Makes a purchase with a credit card, debit card, or gift card. SALE_AUTH : Verifies/authorizes a payment amount on a credit card. CAPTURE : Places a SALE_AUTH (Authorization only) transaction into an open credit card batch. CAPTURE_ALL : Performs a settlement or batch close. REFUND : Returns a credit card, debit card, or gift card payment from a settled batch. VOID : Removes a credit card or gift card transaction from an unsettled batch. REVERSAL : Removes a credit card transaction from an unsettled batch in 'real time'. BALANCEINQUIRY : Performs an inquiry for the remaining card balance against a gift card. ACTIVATE : Performs an activation and sets the initial balance of a new gift card. REACTIVATE : Adds funds to an existing balance of an active gift card. DEACTIVATE : Permanently terminates an active gift card.	String; Min Value = 4; Max Value = 25
Amount	Required. Total transaction amount (includes subtotal, cash back, tax, and tip (Format example: DDDD.CC))	String; Min Value = 4; Max Value = 9
Username	Required. Username of the Merchant	String; Min Value = 5; Max Value = 25
Password	Required. Password of the Merchant.	String; Min Value = 7; Max Value = 25
Zip	Reserved for future use.	String; Min Value = 5; Max Value = 9
Street	Reserved for future use.	String; Min Value = 4;

		Max Value = 25
OrigRefNum	Original reference number. Used for follow-on transactions (e.g., Void, Reversal).	String; Min Value = 1; Max Value = 15
Country	Reserved for future use.	String; Min Value = 2; Max Value = 25
Phone	Reserved for future use.	String; Min Value = 10; Max Value = 25
Email	Reserved for future use.	N/A
MerchantCode	Required. BridgePay Merchant Code.	String; Min Value = 1; Max Value = 9
MerchantAccountCode	Required. BridgePay Merchant Account Code.	String; Min Value = 1; Max Value = 9
InvNum	Required. POS system invoice/tracking number.	String; Min Value = 1; Max Value = 12
ReferenceID	Optional. Can be populated to echo back a variable in the response message.	String; Min Value = 1; Max Value = 50
TerminalType	Required. Terminal device type to be used for transaction processing. Valid values are (case-sensitive): keyed : Keyed transactions, will show the card entry screen. rbabt : Ingenico Bluetooth (MSR and EMV supported) rbausb : Ingenico USB (MSR and EMV supported) unipayiii : IDTech UniPay III (MSR supported; EMV support coming soon in a future build) chipper2 : BBPOS Chipper 2 via audio jack (MSR and EMV supported) wisepad2 : BBPOS WisePad 2 via Bluetooth (MSR and EMV supported)	String; Min Value = 1; Max Value = 25
PartialAuthorization	Optional. Sets the partial authorization flag for a transaction. The default setting is 'false' (meaning not active).	String; Min Value = 4; Max Value = 5
CardholderName	Optional. Name as it appears on the credit card, debit card, or gift card.	String; Min Value = 1; Max Value = 25
CardNumber	Optional. Account number of a credit card, debit card, or gift card used when collecting card data outside of PayGuardian (not recommended).	String; Min Value = 1; Max Value = 19
ExpDate	Optional. Expiration date of a credit card, debit card, or gift card (required for Token transactions).	String; Min Value = 4; Max Value = 4

CvvNumber	Optional. Card verification code of a credit card, debit card, or gift card used when collecting card data outside of PayGuardian (not recommended).	String; Min Value = 3; Max Value = 4
-----------	--	--

3.1.5. Program Output Object

The following table contains descriptions for the PaymentResponse output object properties.

⚡ You should only look at the results in the PaymentResponse

Property	Description	Data Type / Min - Max Values
AuthCode	Transaction authorization code from the payment processor. This value is an approval code for approved transactions or an error code/message for declined transactions.	String; Max Value = 15
ApprovedAmount	The actual amount approved by host. This may differ from the requested amount.	String; Max Value = 15
AvsResponse	AVS response. See AVS Response Codes on pages 35-36 for a list of possible responses.	String; Max Value = 10
BogusAccountNumber	Partially masked card number. The first 6 and last 4 digits of the card number are present with zeros in between. This card number will not pass the mod 10 check.	String; Max Value = 20
CardType	Displays the card type (determined by BIN range).	String; Max Value = 20
CvResponse	The CV response code. See CV Response Codes on pages 36 for a list of possible responses.	String; Max Value = 10
RefNum	Gateway reference/PnRef number. Used for follow-on transactions (i.e., Void, Reversal).	String; Max Value = 15
RemainingBalance	Remaining balance on gift or prepaid card.	String; Max Value = 15
RequestedAmount	Original requested amount of the transaction.	String; Max Value = 15
ResultCode	Result code of the transaction. See on page 35 for more information.	String; Max Value = 10
ResultTxt	Details from the processor or payment gateway about the transaction result.	String; Max Value = 15
Timestamp	Time and date of the transaction.	String;

		Max Value = 30
ExpirationDate	Echo back of the expiration date.	String; Max Value = 4
GatewayMessage	Message from the gateway.	String; Max Value = 30
InternalMessage	Detailed information provided by the gateway / processor regarding results of the transaction request.	String; Max Value = 50
AVSMessage	Unipay AVS Match Result Message.	String; Max Value = 10
CVMMessage	Unipay CVV/CVV2 Match Result Message.	String; Max Value = 10
IsoCountryCode	Country code from the gateway.	String; Max Value = 10
IsoRequestDate	Requested date from the gateway.	String; Max Value = 10
NetworkReferenceNumber	Gateway Network reference number.	String; Max Value = 15
MerchantCategoryCode	Merchant Category Code from the gateway.	N/A
NetworkMerchantId	Merchant id from the gateway.	N/A
NetworkTerminalId	Network terminal id from the gateway.	N/A
ResponseTypeDescription	Reserved for future use.	N/A
StreetMatchMessage	Reserved for future use.	N/A
ExtData	Returns extra data for the extended data fields submitted in the processed transaction (multiple field information may be returned in XML format).	N/A
Token	Represents the tokenized card number received from a token request or authorization.	String; Max Value = 22

3.1.6. PayGuardian Android XML Data Format

Familiarity of the DLL interface, its properties and its uses is must, and it is the basis of the PayGuardian Android XML format. A request contains a PaymentRequest root tag and nested data elements containing DLL properties as tags. The response XML has a similar framework containing all of the standard response data.

3.1.6.1. Examples

Basic transaction request / response and receipt output examples.

MSR Sale Type Request

An example of a basic MSR sale transaction request.

```
payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB);
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("5.00");
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType(PayGuardian.TransactionTypes.SALE.getType());

boolean started = payGuardian.processTransaction(paymentRequest, true);
```

MSR Sale Type Response

An example of a basic MSR sale transaction response.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}

response = {PaymentResponse@21368}
AVSMessage = null
CVMessage = null
approvedAmount = {String@21410} "5.00"
authCode = {String@21411} "101400"
avsResponse = null
bogusAccountNumber = {String@21412} "*****9269"
cardType = null
cashBackAmount = null
cvResponse = null
expirationDate = {String@21413} "0325"
extData = null
gatewayMessage = {String@21414} "A01 - Approved"
gatewayResult = null
hostCode = null
hostResponse = null
internalMessage = {String@21415} "Approved: 101400 (approval code)"
isCommercialCard = {String@21416} "False"
isoCountryCode = null
```

```

isoCurrencyCode = null
isoRequestDate = null
isoTransactionDate = null
merchantCategoryCode = null
message = null
message1 = null
message2 = null
networkMerchantId = null
networkReferenceNumber = null
networkTerminalId = null
rawResponse = null
refNum = {String@21417} "217228301"
remainingAmount = {String@21418} "0.00"
remainingBalance = null
requestedAmount = null
responseTypeDescription = null
resultCode = {String@21419} "00000"
resultTxt = {String@21420} "Successful Request"
streetMatchMessage = null
submittedAmount = {String@21421} "5.00"
timeStamp = {String@21422} "20161230"
tipAmount = null
token = {String@21423} "11110000000077149269"
transactionCode = null

```

MSR Sale Receipt

An example of the values returned on a basic MSR Sale receipt.

BPNPayment.BPNReceipt:

```

maskedCardNumber: 5*****0608:
chipCardAID: (null) //← No AID for non-emv trx
invoice: 8888
seq: 8888
authorizationCode: 290144
entryMethod: Swipe_Read
totalAmount: 5.5
appLabel:
cardHolderName: (null)
networkMerchantId: 518564010126944
networkTerminalId: PPB01.
cardFirstFour: 5413
cardType: Mastercard
requiresSignature: YES
pinEntered: NO

```

Sale_Auth Transaction Type Request

An example of a basic authorization only transaction request.

```
payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB);
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("5.00");
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType(PayGuardian.TransactionTypes.SALE_AUTH.getType());

boolean started = payGuardian.processTransaction(paymentRequest, true);
```

Sale_Auth Transaction Type Response

An example of a basic authorization only transaction response.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}

response = {PaymentResponse@21365}
  AVSMessage = null
  CVMMessage = null
  approvedAmount = {String@21406} "5.00"
  authCode = {String@21407} "842171"
  avsResponse = null
  bogusAccountNumber = {String@21408} "*****9269"
  cardType = null
  cashBackAmount = null
  cvResponse = null
  expirationDate = {String@21409} "0325"
  extData = null
  gatewayMessage = {String@21410} "A01 - Approved"
  gatewayResult = null
  hostCode = null
  hostResponse = null
  internalMessage = {String@21411} "Approved: 842171 (approval code)"
  isCommercialCard = {String@21412} "False"
  isoCountryCode = null
  isoCurrencyCode = null
  isoRequestDate = null
```

```

isoTransactionDate = null
merchantCategoryCode = null
message = null
message1 = null
message2 = null
networkMerchantId = null
networkReferenceNumber = null
networkTerminalId = null
rawResponse = null
refNum = {String@21413} "217228401"
remainingAmount = {String@21414} "0.00"
remainingBalance = null
requestedAmount = null
responseTypeDescription = null
resultCode = {String@21415} "00000"
resultTxt = {String@21416} "Successful Request"
streetMatchMessage = null
submittedAmount = {String@21417} "5.00"
timeStamp = {String@21418} "20161230"
tipAmount = null
token = {String@21419} "11110000000077149269"
transactionCode = null

```

Sale_Auth Receipt

An example of the values returned on a basic authorization only receipt.

BPNPayment.BPNReceipt:

```

maskedCardNumber: 4*****9269:
chipCardAID: A0000000031010
invoice: 33333
seq: 33333
authorizationCode: 786986
entryMethod: Chip_Read
totalAmount: 7.5
appLabel: VISA DEBIT
cardHolderName: CARDHOLDER/VALUED
networkMerchantId: 518564010126944
networkTerminalId: PPB01.
cardFirstFour: 4204
cardType: Visa
requiresSignature: YES
pinEntered: NO

```

Capture Transaction Type with Tip Request

An example of a basic capture transaction request that includes a tip amount in the summary total.

```
payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB);
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("7.00"); //added 2.00 for tip from previous SALE_AUTH trx
paymentRequest.setPnRefNum("217228401");
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType(PayGuardian.TransactionTypes.CAPTURE.getType());

boolean started = payGuardian.processTransaction(paymentRequest, true);
```

Capture Transaction Type with Tip Response

An example of a basic capture transaction response that includes a tip amount in the summary total.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}

response = {PaymentResponse@21264}
  AVSMessage = null
  CVMessage = null
  approvedAmount = null
  authCode = null
  avsResponse = null
  bogusAccountNumber = null
  cardType = null
  cashBackAmount = null
  cvResponse = null
  expirationDate = null
  extData = null
  gatewayMessage = {String@21312} "A01 - Approved"
  gatewayResult = null
  hostCode = null
  hostResponse = null
  internalMessage = null
```



```

isCommercialCard = null
isoCountryCode = null
isoCurrencyCode = null
isoRequestDate = null
isoTransactionDate = null
merchantCategoryCode = null
message = null
message1 = null
message2 = null
networkMerchantId = null
networkReferenceNumber = null
networkTerminalId = null
rawResponse = null
refNum = {String@21313} "217228401"
remainingAmount = null
remainingBalance = null
requestedAmount = null
responseTypeDescription = null
resultCode = {String@21314} "00000"
resultTxt = {String@21315} "Successful Request"
streetMatchMessage = null
submittedAmount = null
timeStamp = null
tipAmount = null
token = null
transactionCode = null

```

Capture Transaction with Tip Receipt

An example of the values returned on a basic capture transaction receipt that includes a tip amount in the summary total.

BPNPayment.BPNReceipt:

```

maskedCardNumber:  :
chipCardAID:      (null)
invoice: 33333
seq: 33333
authorizationCode: (null)
entryMethod: (null)
totalAmount: (null)
appLabel:
cardHolderName: (null)
networkMerchantId: (null)
networkTerminalId: (null)
cardFirstFour: (null)
cardType: (null)
requiresSignature: NO
pinEntered: NO

```

Sale Transaction Type Request using a Token

An example of a basic sale transaction using a token to replace the payment card.

```
payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("6.00");
paymentRequest.setToken("11110000000077149269"); //token from previous request
paymentRequest.setExpDate("0325"); //exp date from previous request
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType(PayGuardian.TransactionTypes.SALE.getType());

boolean started = payGuardian.processTransaction(paymentRequest, true);
```

Sale using Token Transaction Response

An example of a basic sale transaction response where a token was submitted in the transaction request.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}

response = {PaymentResponse@21262}
AVSMessage = null
CVMessage = null
approvedAmount = {String@21308} "6.00"
authCode = {String@21309} "745331"
avsResponse = null
bogusAccountNumber = {String@21310} "*****9269"
cardType = null
cashBackAmount = null
cvResponse = null
expirationDate = {String@21311} "0325"
extData = null
gatewayMessage = {String@21312} "A01 - Approved"
gatewayResult = null
hostCode = null
hostResponse = null
internalMessage = {String@21313} "Approved: 745331 (approval code)"
isCommercialCard = {String@21314} "False"
```

```

isoCountryCode = null
isoCurrencyCode = null
isoRequestDate = null
isoTransactionDate = null
merchantCategoryCode = null
message = null
message1 = null
message2 = null
networkMerchantId = null
networkReferenceNumber = null
networkTerminalId = null
rawResponse = null
refNum = {String@21315} "217228501"
remainingAmount = {String@21316} "0.00"
remainingBalance = null
requestedAmount = null
responseTypeDescription = null
resultCode = {String@21317} "00000"
resultTxt = {String@21318} "Successful Request"
streetMatchMessage = null
submittedAmount = {String@21319} "6.00"
timeStamp = {String@21320} "20161230"
tipAmount = null
token = {String@21321} "11110000000077149269"
transactionCode = null
shadow$_klass_ = {Class@21257} "class
    com.bpn.payguardian.android.model.PaymentResponse"
shadow$_monitor_ = -1712051106

```

Sale using Token Receipt

An example of the values returned on a basic sale transaction where a token was submitted in the transaction request.

BPNPayment.BPNReceipt:

```

maskedCardNumber:  :
chipCardAID:   (null)
invoice: 33333
seq:   33333
authorizationCode:   099912
entryMethod: (null)
totalAmount:   8
appLabel:
cardHolderName:   (null)
networkMerchantId: 518564010126944
networkTerminalId:  PPB01.
cardFirstFour: (null)
cardType:
requiresSignature:   NO
pinEntered:   NO

```

Invalid Tender Type Request

An example of how to submit a sale transaction with an unacceptable tender type.

```
payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB);
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("6.00");
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType("SOME INVALID TYPE");

boolean started = payGuardian.processTransaction(paymentRequest, true);
```

Invalid Tender Type Response

An example of a sale transaction response error where an unacceptable tender type was processed.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}

response = {PaymentResponse@21211}
AVSMessage = null
CVMessage = null
approvedAmount = null
authCode = null
avsResponse = null
bogusAccountNumber = null
cardType = null
cashBackAmount = null
cvResponse = null
expirationDate = null
extData = null
gatewayMessage = null
gatewayResult = null
hostCode = null
hostResponse = null
internalMessage = null
isCommercialCard = null
isoCountryCode = null
isoCurrencyCode = null
isoRequestDate = null
```

```

isoTransactionDate = null
merchantCategoryCode = null
message = null
message1 = null
message2 = null
networkMerchantId = null
networkReferenceNumber = null
networkTerminalId = null
rawResponse = null
refNum = null
remainingAmount = null
remainingBalance = null
requestedAmount = null
responseTypeDescription = null
resultCode = {String@21254} "E0008"
resultTxt = {String@21255} "Invalid transaction type."
streetMatchMessage = null
submittedAmount = null
timeStamp = null
tipAmount = null
token = null
transactionCode = null

```

Cancel Transaction Request

An example of how to submit a sale transaction to simulate the cancellation of a transaction after the transaction has been initiated.

```

payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB);
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("6.00");
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType(PayGuardian.TransactionTypes.SALE.getType());

boolean started = payGuardian.processTransaction(paymentRequest, true);

```

Cancel Transaction Error Response

An example of a cancelled sale transaction error response where the cancellation of a transaction occurred after the transaction was initiated.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}
response = {PaymentResponse@21255}
AVSMessage = null
CVMessage = null
approvedAmount = null
authCode = null
avsResponse = null
bogusAccountNumber = null
cardType = null
cashBackAmount = null
cvResponse = null
expirationDate = null
extData = null
gatewayMessage = null
gatewayResult = null
hostCode = null
hostResponse = null
internalMessage = null
isCommercialCard = null
isoCountryCode = null
isoCurrencyCode = null
isoRequestDate = null
isoTransactionDate = null
merchantCategoryCode = null
message = null
message1 = null
message2 = null
networkMerchantId = null
networkReferenceNumber = null
networkTerminalId = null
rawResponse = null
refNum = null
remainingAmount = null
remainingBalance = null
requestedAmount = null
responseTypeDescription = null
resultCode = {String@21302} "Transaction cancelled"
resultTxt = {String@21302} "Transaction cancelled"
streetMatchMessage = null
submittedAmount = null
timeStamp = null
tipAmount = null
token = null
transactionCode = null
```

3.2. EMV Transaction Support

3.2.1. EMV Implementation Details

PayGuardian Android also supports EMV transactions. Upon receiving the transaction request, PayGuardian Android calls the device to obtain the card information. The device will prompt to Swipe or Insert the Card. Upon the card insertion or card swipe, the transaction will be processed according to the user selection.

Minimum Requirement:

Card Reader Version: 15.0.4 RBA

When a transaction is being processed as an EMV transaction, the communication messages between the Card Reader device and the PayGuardian Android Application are uniquely identified by a "33." message identifier. There are currently seven message types used during EMV transactions. The message type is selected using a subcommand identifier which is embedded in the message. Subcommand identifiers are used to identify the different message types:

1. EMV Transaction Initiation Message

The '00.' EMV Transaction Initiation message is sent from the PayGuardian to the device to indicate the type of transaction purchase.

2. EMV Track 2 Equivalent Data Message

The '02.' EMV Track 2 Equivalent Data message is sent from the device to the PayGuardian. This data is similar to the Track 2 data which is stored on a magnetic stripe card.

3. EMV Authorization Request Message

The '03.' EMV Authorization Request message is sent from the device to the PayGuardian to provide the cryptographic information necessary to authorize the transaction. The authorization process is initiated by the device issuing a request to the PayGuardian.

4. EMV Authorization Response Message

The '04.' EMV Authorization Response message is sent from the PayGuardian to the device in response to the EMV Authorization Request message. This message includes cryptographic information which is read by the embedded microchip on the card.

5. EMV Authorization Confirmation Response Message

The '05.' EMV Confirmation Response message is sent from the device to the PayGuardian, and contains the results from applying the authorization data to the embedded microchip on the EMV card.

3.2.2. EMV Purchase Transaction Flow

This section describes the message flow for EMV transactions once PayGuardian receives the transaction request from the mPOS.

- From the Credit or Debit screen, enter the transaction amount and Invoice Number, then click the Process button.
- The Process Secure Payments window will display and prompt to insert or swipe a card on the Device.
- On the Device, select the language, then confirm the transaction amount.
- The card number, expiration date, and cardholder name fields will now be populated in the Process Secure Payments window from the Device. Click the Process Securely button.
- The transaction response will be returned to PayGuardian Android and to the Device. If the Device has a signature capture pad, Sign and Accept on the Device.
- The electronic signature will display on screen in PayGuardian Android. Click Accept.
- Remove the EMV card from the Device.

Sample EMV Sale Request:

An example of a basic EMV sale transaction request.

```
payGuardian = new com.bpn.payguardian.android.PayGuardian(getApplicationContext(),
    this, this);

PaymentRequest paymentRequest = new PaymentRequest();
paymentRequest.setTerminalType(PayGuardian.TERMINAL_TYPE_RBAUSB);
paymentRequest.setUsername("USERNAME");
paymentRequest.setPassword("PASSWORD");
paymentRequest.setMerchantCode("MERCHANT_CODE");
paymentRequest.setMerchantAccountCode("MERCHANT_ACCOUNT_CODE");
paymentRequest.setInvNum("1234");
paymentRequest.setAmount("5.00");
paymentRequest.setTenderType(PayGuardian.TenderTypes.CREDIT.getType());
paymentRequest.setTransType(PayGuardian.TransactionTypes.SALE.getType());

boolean started = payGuardian.processTransaction(paymentRequest, true);
```

Sample EMV Sale Response:

An example of a basic EMV sale transaction response.

```
@Override
public void transactionResponse(PaymentResponse response)
{
    //handle response
}
```



```

}

response = {PaymentResponse@21368}
  AVSMessage = null
  CVMMessage = null
  approvedAmount = {String@21410} "5.00"
  authCode = {String@21411} "101400"
  avsResponse = null
  bogusAccountNumber = {String@21412} "*****9269"
  cardType = null
  cashBackAmount = null
  cvResponse = null
  expirationDate = {String@21413} "0325"
  extData = null
  gatewayMessage = {String@21414} "A01 - Approved"
  gatewayResult = null
  hostCode = null
  hostResponse = null
  internalMessage = {String@21415} "Approved: 101400 (approval code)"
  isCommercialCard = {String@21416} "False"
  isoCountryCode = null
  isoCurrencyCode = null
  isoRequestDate = null
  isoTransactionDate = null
  merchantCategoryCode = null
  message = null
  message1 = null
  message2 = null
  networkMerchantId = null
  networkReferenceNumber = null
  networkTerminalId = null
  rawResponse = null
  refNum = {String@21417} "217228301"
  remainingAmount = {String@21418} "0.00"
  remainingBalance = null
  requestedAmount = null
  responseTypeDescription = null
  resultCode = {String@21419} "00000"
  resultTxt = {String@21420} "Successful Request"
  streetMatchMessage = null
  submittedAmount = {String@21421} "5.00"
  timeStamp = {String@21422} "20161230"
  tipAmount = null
  token = {String@21423} "11110000000077149269"
  transactionCode = null

```

Sample EMV Sale Receipt

An example of the values returned on a basic EMV Sale receipt.

BPNPayment.BPNReceipt:

maskedCardNumber: 4*****9269:

chipCardAID: A0000000031010

invoice: 12345

seq: 12345

authorizationCode: 471130

entryMethod: Chip_Read

totalAmount: 6

appLabel: VISA DEBIT

cardHolderName: CARDHOLDER/VALUED

networkMerchantId: 518564010126944

networkTerminalId: PPB01.

cardFirstFour: 4204

cardType: Visa

requiresSignature: YES

pinEntered: NO

A. Appendix

A.1. Response Values

A.1.1. Result Codes

The following table contains descriptions of the result codes returned in the **ResultCode** response field from PayGuardian. Please note that when programmatically validating the result of a transaction, you should use this value instead of any other response message describing the result.

Value	Description
0	Approved.
1	Cancelled. Review ResultTxt field in PaymentResponse to determine why the transaction was not processed.
2	Declined. Review ResultTxt field in PaymentResponse to determine why the transaction was not processed.

A.1.2. AVS Response Codes

The following table contains the possible descriptions of the response values returned for Address Verification (AVS) in the **AvsResponse** response field from PayGuardian.

Please Note: If the response returned is blank for this specific field tag, it is possible that your selected processor does not support the full range of AVS codes.

Value	Description
00	AVS Error – Retry, System unavailable, or Timed out
40	Address not available (Address not verified)
43	Street address not available (not verified), ZIP matches
44	Address failed
45	Street address and Zip don't match
46	Street address doesn't match, 5-digit ZIP matches
47	Street address doesn't match, 9-digit ZIP matches

4A	Street address or ZIP doesn't match
4D	Street address matches, ZIP does not
4E	Street address and 5-digit ZIP matches
4F	Street address and ZIP match
53	Account holder name incorrect, billing postal code matches
55	Unrecognized response – Account holder name, billing address, and postal code are all incorrect
5C	Account holder name incorrect, billing address matches
5F	Account holder name incorrect, billing address and postal code match
70	Account name matches
73	Account holder name and billing postal code match
7C	Account holder name and billing address match
7F	Account holder name, billing address and postal code match
80	AVS service not supported by issuer – Issuer doesn't participate in AVS

A.1.3. CV Response Codes

The following table contains the possible descriptions of the response values returned for a CVV2/CVC2/CID check in the **CvResponse** response field from PayGuardian.

Please Note: If the response returned is blank for this specific field tag, it is possible that your selected processor does not support the full range of CVV response codes.

Value	Description
M	CVV2/CVC2/CID – Match.
N	CVV2/CVC2/CID – No Match.
P	Not Processed.
S	Issuer indicates that the CV data should be present on the card, but the merchant has indicated that the CV data is not present on the card.
U	Unknown. Issuer has not certified for CV or issuer has not provided Visa/MasterCard with the CV encryption keys.
X	Unrecognized reason. Server provided did not respond.

A.2. Errors

A.2.1. App to App Communication Errors

Result Code	Description
E1001	Invalid invoice number
E1002	Invalid amount.
E1003	Invalid username.
E1004	Invalid password.
E1005	Invalid merchant code.
E1006	Invalid merchant account code.
E1007	Invalid tender type.
E1008	Invalid transaction type.
E1009	Invalid payment request
E1010	Invalid reference number.
E1011	No network connection.
E1012	Device not found.
E1013	Xml deserialization error.
E1014	Xml serialization error
E1015	Xml base64 encode error.
E1016	Xml base64 decode error.
E1017	Unable to process transaction
E1018	Unable to access device.
E1019	Transaction cancelled, Device TimedOut.
E1020	Transaction cancelled.

A.3. Test Cards and Data

A.3.1. Swiped / MSR Test Transaction Info

The following test card numbers are required for use in both integration testing and certification for PayGuardian Android, but only for **'NON-EMV TRANSACTIONS'** (a separate test account will be provided for EMV transaction testing):

Test Cards & Bank Accounts

The following accounts will be accepted by the test server's validation mechanism and thus can be used for preliminary testing:

Card Type	Card Brand	Card Number	Expiration Date	Track Data
Credit	Visa	4111111111111111	1017	%B4111111111111111^Smith/John^160410110001111A123456789012?
Credit	MasterCard	5499740000000057	1017	%B5499740000000057^Smith/John^160410110001111A123456789012?
Credit	Discover	6011000991001201	1017	%B6011000991001201^Smith/John^160410110001111A123456789012?
Credit	Amex	371449635392376	1017	%B371449635392376^Smith/John^160410110001111A123456789012?
Debit	Visa	4217651111111119	1017	%B4217651111111119^Smith/John^16041011234567440?;4217651111111119=16041011234567440?
Debit	MasterCard	5149612222222229	1017	%B5149612222222229^Smith/John^16041011234567440?;5149612222222229=16041011234567440?

Encrypted Data

Card Number	Expiration Date	Secure Format	MSRKSN	Track 1 Encrypted	Track 2 Encrypted
5526399000648568	0416	SecureMagV2	310601012B000 B40007B	BD12FA1B13FB19BD889 7CBCF4109205B69133A 78F2B2181F2D3849B48 4303F4CEB54FD285774 8B0D92E72E0BAFB9D9 876C431FF24F37128665 50DA1D697E84F353F3E EBB8AA42E4F	67E98B7924544F5803 BC4A4453834BD13D1 929CD7006360A551A 1890383011EBBEDBB 3442BD71F45
4012000033330026	0416	SecureMagV2	310601012B000 B400080	AA06168B0C7DFBF22D FEC13D7B49242906D26 39C65E4AAF5364B143E D5DCB9F45AA0BA73F 631032B83648311E88E4 ED921451F0E9684E866 613F35BD74F53802C3A A9E56643863E988DD4B ABD4753A5	D011E8820440DE01C D626AE55921ADA5F3 D475EB9150E2959DF A1EE6353A3204EF7F 69797B97D753
5473500000000014	1225	SecureMag	9107010000000 0800007	EA02EB27DAFC3BE1AE 43C318D96F39E9AB90D 55B0978614D7860C8957 21DD0E9EA358FDDCA4 A0FAADE3D1E0E69D91 FCB31B70E73B8FB5E49 F4E4AF6219AB9B6F75C FFF8A6385DE6645426E C7E9DDD128C7D63E66 2B2C6E969E3CEE75789 F66C48251B69A6B9C79 CDC570FD984867ACA2	
4012002000060016	1225	SecureMag	9107010000000 0800008	C14C88046A89D221FA5 0EF0DE61B61C4C11309 5BB6FB2D6185BC5100A BB0ABF6DDA5F7D1F4F 3DFA16DFB365EA098C C927195B4A3F9762BF8 5262D1530C88181B794 E3CADBEA469539DEBB D341751599322B1A7A4 D2C760044CCC2B311B 4D175959FCE0E2DF9B0 BB7	

Test amount ranges

As a part of the testing, the amount ranges specified below, can be used to trigger specific response codes from the server. Any valid account number and any properly formatted billing address can be used for the test.

Sales and Authorizations

Amount Range		Credit Cards
In dollars	In cents	Response Message
5.00 – 69.99	500 – 6999	Approved.
70.00 – 79.99	7000 – 7999	Invalid Card Number (Invalid Account Number)
80.00 – 89.99	8000 – 8999	Card reported lost/stolen (Lost/Stolen Card)
90.00 – 99.99	9000 – 9999	Call for Authorization (Referral)*
100.00 – 109.99	10000 – 10999	Hold – Pick up card (Pick Up Card)
110.00 – 119.99	11000 – 11999	CSC is invalid (Decline CSC/CID Fail)
120.00 – 129.99	12000 – 12999	Insufficient Funds
130.00 – 139.99	13000 – 13999	Processing Network Unavailable
140.00 – 149.99	14000 – 14999	Processing Network Error
150.00 – 159.99	15000 – 15999	Partially Approved**

*This range is designated to test voice authorization. Sale, Auth request in the amount between \$90.00 – 99.99 will Response in a decline code, however if approval code 012345 is specified, then an approval will be received.

**This range is designated to test partial authorizations. By setting Sale with this amount range the partially approved transaction will be received. Approved amount will be \$10 less than the originally requested amount.

Credits

Amount Range (in dollars)	Amount Range (in cents)	Response Message
5.00 – 69.99	500 - 6999	Credit Posted

A.4. Terms of Use Agreement

1. ACKNOWLEDGMENT AND ACCEPTANCE OF AGREEMENT

The Terms of Use Agreement "TOU" is provided by BridgePay to you as an end user "USER" of the information obtained from BridgePay, any amendments thereto, and any operating rules or policies that may be published from time to time by BridgePay, all of which are hereby incorporated by reference. The TOU comprises the entire agreement between USER and BridgePay and supersedes any prior agreements pertaining to the subject matter contained herein.

2. DESCRIPTION OF SPECIFICATIONS AND INFORMATION

BridgePay is providing USER with the information concerning the technical requirements or allowing point of sale software to send and receive electronic transaction data to the BridgePay network for authorization and/or settlement. To utilize the Specifications, USER must: (i) provide for USER's own access to the World Wide Web and pay any fees associated with such access, and (ii) provide all equipment necessary for USER to make such connection to the World Wide Web, including a computer, modem and Web browser.

3. USER'S REGISTRATION OBLIGATIONS

In consideration of use of the Specifications, USER agrees to: (i) provide true, accurate, current, and complete information about USER as requested on the Registration Form, and (ii) to maintain and update this information to keep it true, accurate, current and complete. This information about a USER shall be referred to as "Registration Data". If any information provided by USER is untrue, inaccurate, not current, or incomplete, BridgePay has the right to terminate USER's access to the Specifications and refuse any and all current or future use of the Specifications.

4. MODIFICATIONS TO AGREEMENT

BridgePay may change the TOU from time to time at its sole discretion. Changes to the TOU will be announced and publicly available to all USERS.

5. MODIFICATIONS TO SPECIFICATIONS

BridgePay reserves the right to modify or discontinue, temporarily or permanently, the use of any of the Specifications with or without notice to USER. USER agrees that BridgePay shall not be liable to USER or any third party for any modification or discontinuance of a Specification.

6. USER ACCOUNT, PASSWORD AND SECURITY

USER will receive a password when registering their company (account) to become a Partner. Upon approval, that password will allow USER access into the Partner Portal. USER is responsible for maintaining the confidentiality of the password and account, and is fully responsible for all activities that occur under USER's password or account. USER agrees to immediately notify BridgePay of any unauthorized use of USER's password or account or any other breach of security.

7. LICENSE GRANT

- a. Subject to the terms and conditions of this Agreement, BridgePay hereby grants to USER a personal, limited, perpetual, non-exclusive, non-transferable, annual

subscription license to make and use the SOFTWARE accompanying this TOU to be installed on CPUs residing on Licensee's premises, solely for Licensee's internal use. BridgePay and its suppliers shall retain title and all ownership rights to the product and this Agreement shall not be construed in any manner as transferring any rights of ownership or license to the SOFTWARE or to the features or information therein, except as specifically stated herein. use and reproduce the following solely to develop, manufacture, test and support the products: (i) in object code form (except as may be agreed by the parties in writing or as otherwise set forth in this Agreement), (ii) the applicable software in object code form, except as may be agreed by the parties in writing or as otherwise set forth in this Agreement, (iii) BridgePay materials which shall include all documentation.

- b. Upon the annual anniversary date of the activation of the license, the USER will be responsible the renewal of the subscription of the license either through their RESELLER or directly to BridgePay.

8. TRADEMARKS

USER acknowledges that BridgePay owns exclusive rights in the BridgePay trademarks. USER will not use BridgePay as part of any of its product, service, domain or company names and will not take nor authorize any action inconsistent with BridgePay's' exclusive trademark rights during the term of this Agreement or thereafter. Nothing in this Agreement grants USER ownership or any rights in or to use the BridgePay trademarks except in accordance with this license. USER will use a legend on its website and, where commercially feasible, on all printed materials and products bearing the BridgePay trademarks similar to the following: "(USER name) uses the BridgePay™ mark under express license from BridgePay, LLC."

9. USER OBLIGATIONS

- a. USER shall utilize its BridgePay assigned developer ID in each application utilizing the BridgePay specification
- b. USER shall not reverse-engineer, reverse-compile or disassemble any BridgePay software or otherwise attempt to derive the source code to any BridgePay software.
- c. USER shall have no right to (i) disclose any BridgePay source code or BridgePay source code or BridgePay source code documentation other than as permitted or contemplated by this Agreement. No licenses are granted by BridgePay to USER by implication or estoppels to the BridgePay source code or BridgePay source code documentation.
- d. USER shall comply with all applicable card association regulations, applicable federal, state and local statutes and BridgePay required procedures and identified best practices. USER agrees (i) not to use the Specifications for illegal purposes; and (ii) to comply with all applicable laws regarding the transmission of technical data exported from the United States.

10. DISCLAIMER OF WARRANTIES

USER expressly agrees that use of the Specifications is at USER's sole risk. The Specifications are provided on an "as is" basis.

- a. BRIDGEPAY EXPRESSLY DISCLAIMS ALL WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT
- b. BRIDGEPAY MAKES NO WARRANTY THAT THE SPECIFICATION WILL MEET USER'S REQUIREMENTS, NOR DOES BRIDGEPAY MAKE ANY WARRANTY AS TO THE RESULTS THAT MAY BE OBTAINED FROM THE USE OF THE SPECIFICATIONS OR AS TO THE ACCURACY OR RELIABILITY OF ANY INFORMATION OBTAINED THROUGH USE OF THE SPECIFICATIONS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF CERTAIN WARRANTIES, SO SOME OF THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU.

11. TERMINATION BY BridgePay

USER agrees that BridgePay may terminate USER's password, account or use of the Specifications:

- a. If BridgePay determines in its sole discretion that USER has violated or acted inconsistently with the letter or spirit of the TOU;
- b. If the USER has violated the rights of BridgePay, or that USER's continued use of the Specifications poses a material threat to the security, stability or ongoing operation of the System or Specifications.
- c. BridgePay may terminate this Agreement for cause at any time upon providing not less than ten (10) business day's prior written notice to USER. USER acknowledges and agrees that any termination of access privileges to the Specifications under any provision of the Agreement may be effected without prior notice.
- d. BridgePay shall in the event that a license has not been renewed and BridgePay has not received and validated payment from either the USER or the RESELLER within 30 days of the anniversary date of the original activation the deactivate or terminate the usage of the license.

12. LIMITATION OF LIABILITY

In no event shall BridgePay be liable to USER for any incidental, consequential or punitive damages related to this Agreement or the use of BridgePay software or specifications. The liability of BridgePay hereunder shall be limited to the fees paid to BridgePay pursuant to this Agreement. USER agrees that BridgePay shall not be liable for any direct, indirect, incidental, special, or consequential damages, resulting from the use or the inability to use the Specifications, including but not limited to, damages for loss of profits, use, data or other intangibles, even if BridgePay has been advised of the possibility of such damages. Some jurisdictions do not allow the limitation or exclusion of liability for incidental or consequential damages so some of the above limitations may not apply to you.

NOTICE: Any notice to USER or to BridgePay shall be made via either e-mail or regular mail. BridgePay may also provide notices of changes to the TOU or other matters by displaying notices to USERS, generally on the Specifications.

13. SPECIAL DAMAGES

In no event will BridgePay be liable to the USER, consequential or punitive damages, including but not limited to, lost profits, even if such party knew of the possibility of such damages.

14. INDEMNIFICATION

- a. USER shall be liable to and shall indemnify and hold BridgePay, its employees, representatives, successors and permitted assigns harmless from and against any and all claims, demands by third parties, losses, liability, cost, damage and expense, including litigation expenses and reasonable attorneys' fees and allocated costs for in house legal services, to which BridgePay, its employees, representatives, successors and permitted assigns may be subjected or which it may incur in connection with any claims which arise from or out of or as the result of (i) USER's breach of this Agreement, (ii) the performance by USER of its duties and obligations under this Agreement or (iii) the negligent or willful misconduct of USER, its officers, employees, agents and affiliates in the performance of their duties and obligations under this Agreement.

15. PROTECTION OF CONFIDENTIAL INFORMATION

All information of a business nature relating to the BridgePay specification, software, application interfaces, services, processes, merchant and cardholder data, product or programming techniques of either party shall be deemed confidential ("Confidential Information"). This shall not prohibit each party from disclosing such Confidential Information to persons required to have access thereto for the performance of this Agreement; provided, however, that such persons shall be required to keep such Confidential Information confidential to the same standard that the disclosing party is obligated to keep the Confidential Information confidential.

16. FORCE MAJEURE

In no event shall BridgePay be liable with respect to the failure of its duties and obligations under this Agreement (other than an obligation to pay money) which is attributable to acts of God, war, terrorism, conditions or events of nature, civil disturbances, work stoppages, equipment failures, power failures, fire or other similar events beyond its control.

17. GENERAL

- a. The Specifications Agreement and the relationship between USER and BridgePay shall be governed by the laws of the State of Illinois without regard to its conflict of law provisions.
- b. The failure of BridgePay to exercise or enforce any right or provision of the TOU shall not constitute a waiver of such right or provision. If any provision of the TOU is found by a court of competent jurisdiction to be invalid, the parties nevertheless agree that the court should endeavour to give effect to the parties' intentions as reflected in the provision, and the other provisions of the TOU remain in full force and effect.
- c. USER agrees that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to use of the Specifications or the Specifications

Agreement must be filed within ninety (90) days after such claim or cause of action arose or be forever barred.

18. SECTION TITLES

The section titles in the TOU are for convenience only and have no legal or contractual effect.