WePay Android SDK

1.0.0

Generated by Doxygen 1.8.12

Contents

1 Getting Started

Introduction

The WePay Android SDK enables collection of payments via various payment methods.

It is meant for consumption by WePay partners who are developing their own Android apps aimed at merchants and/or consumers.

Regardless of the payment method used, the SDK will ultimately return a Payment Token, which must be redeemed via a server-to-server API call to complete the transaction.

Payment methods

There are two types of payment methods:

- Consumer payment methods - to be used in apps where consumers directly pay and/or make donations
- Merchant payment methods - to be used in apps where merchants collect payments from their customers

The WePay Android SDK supports the following payment methods:

- EMV Card Reader: Using an EMV Card Reader, a merchant can accept in-person payments by prosessing a consumer's EMV-enabled chip card. Traditional magnetic strip cards can be processed as well.
- Manual Entry (Consumer/Merchant): The Manual Entry payment method lets consumer and merchant apps accept payments by allowing the user to manually enter card info.

Installation

In the following steps, [version] represent one particular sdk version identifier such as 1.0.0 Replace [version] in following steps with the sdk version you are using

- Add the following jars to the libs directory under app directory of your project source:
  1. wepay-android-[version].aar
  2. wepay-android-[version]-javadoc.jar
  3. wepay-android-[version]-sources.jar

  For example, if you are using sdk version 1.0.0, you need to include the following files
  1. wepay-android-1.0.0.aar
  2. wepay-android-1.0.0-javadoc.jar
  3. wepay-android-1.0.0-sources.jar

- Open build.gradle file for you app module (not the build.gradle file of the project) and add the following

```
repositories{
    flatDir{
        dirs 'libs'
    }
}
```

- Also add the following to the dependencies closure

```
compile(name:'wepay-android-[version]', ext:'aar')
compile 'com.google.code.gson:gson:2.2.2'
```

As an example, if you are using sdk version 1.0.0, you need to add the following in dependencies closure

```
compile(name:'wepay-android-1.0.0', ext:'aar')
compile 'com.google.code.gson:gson:2.2.2'
```

- Open your app's manifest.xml and add the following permissions under the manifest tag:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.INTERNET" />
```

- Android 6 / M / API 23 and later require a more complicated mechanism of requesting audio permissions fromt the user. See the WePayExample app's MainActivity.java for a sample implementation.
- Clean and build the project using your IDE or from the command line by going to the project's base directory and running:

```
./gradlew clean build
```

- Done!

Note: Card reader functionality is not available in this SDK by default. If you want to use this SDK with WePay card readers, send an email to mobile@wepay.com.

Documentation

HTML documentation is hosted on our Github Pages Site.

Pdf documentation is available on the releases page or as a direct download.

SDK Organization

com.wepay.android.WePay

The WePay class is the starting point for consuming the SDK, and is the primary class you will interact with. It exposes all the methods you can call to accept payments via the supported payment methods. Detailed reference documentation is available on the reference page for the Wepay class.

Interfaces

The SDK uses interfaces to repond to API calls. You will implement the relevant interfaces to receive responses to the API calls you make. Detailed reference documentation is available on the reference page for each interface:

- com.wepay.android.AuthorizationHandler
- com.wepay.android.CardReaderHandler
- com.wepay.android.CheckoutHandler
- com.wepay.android.TokenizationHandler

Data Models and Enums

All other classes in the SDK are data models and Enums that are used to exchange data between your app and the SDK. Detailed reference documentation is available on the reference page for each class.

Next Steps

Head over to the com.wepay.android.WePay class reference to see all the API methods available. When you are ready, look at the samples below to learn how to interact with the SDK.

Error Handling

com.wepay.android.models.Error serves as documentation for all errors surfaced by the WePay Android SDK.

Samples

See the WePayExample app for a working implementation of all API methods.

Initializing the SDK

- Complete the installation steps (above).
- Include the wepay packages

```
import com.wepay.android.*;
import com.wepay.android.models.*;
import com.wepay.android.enums.*;
```

- Define a property to store the Wepay object

```
WePay wepay;
```

- Create a com.wepay.android.models.Config object

```
String clientId = "your_client_id";
Context context = getApplicationContext();
String environment = Config.ENVIRONMENT_STAGE;

Config config = new Config(context, clientId, environment);
```

- Initialize the WePay object and assign it to the property

```
this.wepay = new WePay(config);
```

(optional) Providing permission to use location services for fraud detection

- Open your app's manifest.xml and add the following permission under the manifest tag:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
```

- Set the option on the config object, before initializing the WePay object

```
config.setUseLocation(true);
```

Integrating the Card Reader payment methods (Swipe+Dip)

- Implement the AuthorizationHandler, CardReaderHandler and TokenizationHandler interfaces

```
public class MainActivity extends ActionBarActivity implements AuthorizationHandler, CardReaderHandler,
      TokenizationHandler
```

- Implement the AuthorizationHandler interface methods

```
@Override
public void onEMVApplicationSelectionRequested(ApplicationSelectionCallback callback, ArrayList<String>
      applications) {
    // Ask the payer to select an application from the list,
    // then execute the callback with the index of the selected application
    callback.useApplicationAtIndex(0);
}

@Override
public void onAuthorizationSuccess(PaymentInfo paymentInfo, AuthorizationInfo authorizationInfo) {
    // Send the tokenId (authorizationInfo.getTokenId()) and transactionToken
       (authorizationInfo.getTransactionToken()) to your server
    // Your server will use these values to make a /checkout/create call to complete the transaction
}

@Override
public void onAuthorizationError(PaymentInfo paymentInfo, Error error) {
    // handle the error
}
```

- Implement the CardReaderHandler interface methods

```
@Override
public void onSuccess(PaymentInfo paymentInfo) {
    // use the payment info (for display/recordkeeping)
    // wait for card tokenization response
}

@Override
public void onError(Error error) {
    // handle the error
}

@Override
public void onStatusChange(CardReaderStatus status) {
    if (status.equals(CardReaderStatus.NOT_CONNECTED)) {
        // show UI that prompts the user to connect the card reader
        this.setStatusText("Connect card reader and wait");
    } else if (status.equals(CardReaderStatus.WAITING_FOR_CARD)) {
        // show UI that prompts the user to swipe/dip
        this.setStatusText("Swipe/Dip card");
    } else if (status.equals(CardReaderStatus.SWIPE_DETECTED)) {
        // provide feedback to the user that a swipe was detected
        this.setStatusText("Swipe detected");
    } else if (status.equals(CardReaderStatus.CARD_DIPPED)) {
        // provide feedback to the user that a dip was detected
        // also let them know they should not remove the card
        this.setStatusText("Card dipped, do not remove card");
    } else if (status.equals(CardReaderStatus.TOKENIZING)) {
        // provide feedback to the user that the card is being tokenized
        this.setStatusText("Tokenizing card...");
```

```
    } else if (status.equals(CardReaderStatus.AUTHORIZING)) {
        // provide feedback to the user that the card is being authorized
        this.setStatusText("Authorizing card...");
    }  else if (status.equals(CardReaderStatus.STOPPED)) {
        // provide feedback to the user that the card reader was stopped
        this.setStatusText("card reader Stopped");
    } else {
        // handle all other status change notifications
        this.setStatusText(status.toString());
    }
}

@Override
public void onReaderResetRequested(CardReaderResetCallback callback) {
    // decide if you want to reset the reader,
    // then execute the callback with the appropriate response
    callback.resetCardReader(false);
}

@Override
public void onTransactionInfoRequested(CardReaderTransactionInfoCallback callback) {
    // provide the amount, currency code and WePay account ID of the merchant
    callback.useTransactionInfo(21.61, CurrencyCode.USD, accountId);
}

@Override
public void onPayerEmailRequested(CardReaderEmailCallback callback) {
    // provide the email address of the payer
    callback.insertPayerEmail("android-example@wepay.com");
}
```

- Implement the TokenizationHandler interface methods

```
@Override
public void onSuccess(PaymentInfo paymentInfo, PaymentToken token) {
    // Send the tokenId (paymentToken.getTokenId()) to your server
    // Your server would use the tokenId to make a /checkout/create call to complete the transaction
}


@Override
public void onError(PaymentInfo paymentInfo, Error error) {
    // Handle error
}
```

- Make the WePay API call, passing in the instance(s) of the class(es) that implemented the interface methods

```
this.wepay.startCardReaderForTokenizing(this, this, this);
// Show UI asking the user to insert the card reader and wait for it to be ready
```

- That's it! The following sequence of events will occur:

  1. The user inserts the card reader (or it is already inserted)
  2. The SDK tries to detect the card reader and initialize it.
     - If the card reader is not detected, the `onStatusChange` method will be called with `status = NOT_CONNECTED`
     - If the card reader is successfully detected, then the `onStatusChange` method will be called with `status = CONNECTED`.
  3. Next, the SDK checks if the card reader is correctly configured (the `onStatusChange` method will be called with `status = CHECKING_READER`).
     - If the card reader is already configured, the App is given a chance to force configuration. The SDK calls the `onReaderResetRequested` method, and the app must execute the callback method, telling the SDK whether or not the reader should be reset.
     - If the reader was not configured, or the app requested a reset, the card reader is configured (the `onStatusChange` method will be called with `status = CONFIGURING_READER`)

4. Next, if the card reader is successfully initialized, the SDK asks the app for transaction information by calling the `onTransactionInfoRequested` method. The app must execute the callback method, telling the SDK what the amount, currency code and merchant account id is.
5. Next, the `onStatusChange` method will be called with `status = WAITING_FOR_CARD`
6. If the user inserts a card successfully, the `onStatusChange:` method will be called with `status = CARD_DIPPED`
7. If the card has multiple applications on it, the payer must choose one:
   - The SDK calls the `onEMVApplicationSelectionRequested` method with a list of Applications on the card.
   - The app must display these Applications to the payer and allow them to choose which application they want to use.
   - Once the payer has decided, the app must inform the SDK of the choice by executing the calback method and passing in the index of the chosen application.
8. Next, the SDK extracts card data from the card.
   - If the SDK is unable to obtain data from the card, the `onError` method will be called with the appropriate error, and processing will stop (the `onStatusChange` method will be called with `status = STOPPED`)
   - Otherwise, the SDK attempts to ask the App for the payer's email by calling the `onPayerEmailRequested` method
9. The app must execute the callback method and pass in the payer's email address.
10. Next, the `onSuccess` method is called with the obtained payment info.
11. Next, the SDK will automatically send the obtained EMV card info to WePay's servers for authorization (the `onStatusChange` method will be called with `status = AUTHORIZING`)
12. If authorization fails, the `onAuthorizationError` method will be called and processing will stop.
13. If authorization succeeds, the `onAuthorizationSuccess` method will be called.
14. Done!

Note: After the card is inserted into the reader, it must not be removed until a successful auth response (or an error) is returned.

Integrating the Manual payment method

- Implement the TokenizationHandler interface

```
public class MainActivity extends ActionBarActivity implements TokenizationHandler
```

- Implement the TokenizationHandler interface methods

```
@Override
public void onSuccess(PaymentInfo paymentInfo, PaymentToken token) {
    // Send the tokenId (paymentToken.getTokenId()) to your server
    // Your server would use the tokenId to make a /checkout/create call to complete the transaction
}


@Override
public void onError(PaymentInfo paymentInfo, Error error) {
    // Handle error
}
```

- Instantiate a PaymentInfo object using the user's credit card and address data

```
Address address = new Address(Locale.getDefault());
address.setAddressLine(0, "380 Portage ave");
address.setLocality("Palo Alto");
address.setPostalCode("94306");
address.setCountryCode("US");

PaymentInfo paymentInfo = new PaymentInfo("Android", "Tester", "a@b.com",
        "Visa xxxx-1234", address,
        address, PaymentMethod.MANUAL,
        "4242424242424242", "123", "01", "18", true);
```

- Make the WePay API call, passing in the instance of the class that implemented the TokenizationHandler interface methods

```
this.wepay.tokenize(paymentInfo, this);
```

- Thats it! The following sequence of events will occur:
  1. The SDK will send the obtained payment info to WePay's servers for tokenization
  2. If the tokenization succeeds, TokenizationHandler's `onSuccess` method will be called
  3. Otherwise, if the tokenization fails, TokenizationHandler's `onError` method will be called with the appropriate error

Integrating the Store Signature API

- Implement the CheckoutHandler interfaces

```
public class MainActivity extends ActionBarActivity implements CheckoutHandler
```

- Implement the CheckoutHandler interface methods

```
@Override
public void onSuccess(String signatureUrl, String checkoutId) {
    // success! nothing to do here
}


@Override
public void onError(Bitmap image, String checkoutId, Error error) {
    // handle the error
}
```

- Obtain the checkout_id associated with this signature from your server

```
String checkoutId = this.obtainCheckoutId();
```

- Instantiate a Bitmap object containing the user's signature

```
Bitmap signature = BitmapFactory.decodeResource(getApplicationContext().getResources(), R.drawable.
      dd_signature);
```

- Make the WePay API call, passing in the instance of the class that implemented the CheckoutHandler interface methods

```
this.wepay.storeSignatureImage(signature, checkoutId, this);
```

- Thats it! The following sequence of events will occur:
  1. The SDK will send the obtained signature to WePay's servers for tokenization
  2. If the operation succeeds, CheckoutHandler's `onSuccess` method will be called
  3. Otherwise, if the operation fails, CheckoutHandler's `onError` method will be called with the appropriate error

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 Class Documentation

3.1 AuthorizationHandler.ApplicationSelectionCallback Interface Reference

Public Member Functions

- void useApplicationAtIndex (int selectedIndex)

3.1.1 Detailed Description

The Interface ApplicationSelectionCallback defines the callback method used to provide information to the card reader during a Dip transaction.

3.1.2 Member Function Documentation

3.1.2.1 void useApplicationAtIndex (int *selectedIndex*)

The callback function that must be executed by the app when onEMVApplicationSelectionRequested() is called by the SDK.

Examples: callback.useApplicationAtIndex(0);

Parameters

selectedIndex	the index of the selected application in the array of applications from the card.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/AuthorizationHandler.java

3.2 AuthorizationHandler Interface Reference

Classes

- interface ApplicationSelectionCallback

Public Member Functions

- void onEMVApplicationSelectionRequested (ApplicationSelectionCallback callback, ArrayList< String > applications)
- void onAuthorizationSuccess (PaymentInfo paymentInfo, AuthorizationInfo authorizationInfo)
- void onAuthorizationError (PaymentInfo paymentInfo, Error error)

3.2.1 Detailed Description

The Interface AuthorizationHandler defines the method used to return data in response to an authorization call.

3.2.2 Member Function Documentation

3.2.2.1 void onAuthorizationError (PaymentInfo *paymentInfo,* Error *error*)

Called when an authorization call fails.

Parameters

paymentInfo	the payment info for the card that failed authorization.
error	the error which caused the failure.

3.2.2.2 void onAuthorizationSuccess (PaymentInfo *paymentInfo,* AuthorizationInfo *authorizationInfo*)

Called when an authorization call succeeds.

Parameters

paymentInfo	the payment info for the card that was authorized.
authorizationInfo	the authorization info for the transaction that was authorized.

3.2.2.3 void onEMVApplicationSelectionRequested (ApplicationSelectionCallback *callback,* ArrayList< String > *applications*)

Called when the EMV card contains more than one application. The applications should be presented to the payer for selection. Once the payer makes a choice, the app must execute callback.useApplicationAtIndex() with the index of the selected application. The transaction cannot proceed until the callback is executed.

Example: callback.useApplicationAtIndex(0);

Parameters

callback	the callback object.
applications	the array of String containing application names from the card.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/Authorization↩ Handler.java

3.3 CardReaderHandler.CardReaderEmailCallback Interface Reference

Public Member Functions

- void insertPayerEmail (String email)

3.3.1 Detailed Description

The Interface CardReaderEmailCallback defines the method used to provide email information to the card reader after a transaction.

3.3.2 Member Function Documentation

3.3.2.1 void insertPayerEmail (String *email*)

The callback function that must be executed by the app when onPayerEmailRequested() is called by the SDK.

Examples: callback.insertPayerEmail("android-example@wepay.com"); callback.insertPayerEmail(null);

Parameters

email	the payer's email address.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/CardReaderHandler.java

3.4 CardReaderHandler Interface Reference

Classes

- interface CardReaderEmailCallback
- interface CardReaderResetCallback
- interface CardReaderTransactionInfoCallback

Public Member Functions

- void onSuccess (PaymentInfo paymentInfo)
- void onError (Error error)
- void onStatusChange (CardReaderStatus status)
- void onReaderResetRequested (CardReaderResetCallback callback)
- void onTransactionInfoRequested (CardReaderTransactionInfoCallback callback)
- void onPayerEmailRequested (CardReaderEmailCallback callback)

3.4.1 Detailed Description

The Interface CardReaderHandler defines the methods used to communicate information regarding the card reader.

3.4.2 Member Function Documentation

3.4.2.1 void onError (Error *error*)

Gets called when the card reader fails to read a card's information.

Parameters

error	the error due to which card reading failed.

3.4.2.2 void onPayerEmailRequested (CardReaderEmailCallback *callback*)

Gets called so that an email address can be provided before a transaction is authorized. The app must respond by executing callback.insertPayerEmail(). The transaction cannot proceed until the callback is executed.

Parameters

callback	the callback object.

3.4.2.3 void onReaderResetRequested (CardReaderResetCallback *callback*)

Gets called when the connected card reader is previously configured, to give the app an opportunity to reset the device. The app must respond by executing callback.resetCardReader(). The transaction cannot proceed until this callback is executed. The card reader must be reset here if the merchant manually resets the reader via the hardware reset button on the reader.

Parameters

callback	the callback object.

3.4.2.4 void onStatusChange (CardReaderStatus *status*)

Gets called whenever the card reader changes status.

Parameters

status	the status.

3.4.2.5 void onSuccess (PaymentInfo *paymentInfo*)

Gets called when the card reader reads a card's information successfully.

Parameters

paymentInfo	the payment info read from a card.

3.4.2.6 void onTransactionInfoRequested (CardReaderTransactionInfoCallback *callback*)

Gets called so that the app can provide the amount, currency code and the WePay account Id of the merchant. The app must respond by executing callback.useTransactionInfo(). The transaction cannot proceed until this callback is executed.

Parameters

callback	the callback object.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/CardReaderHandler.java

3.5 CardReaderHandler.CardReaderResetCallback Interface Reference

Public Member Functions

- void resetCardReader (boolean shouldReset)

3.5.1 Detailed Description

The Interface CardReaderResetCallback defines the method used to provide information to the card reader before a transaction.

3.5.2 Member Function Documentation

3.5.2.1 void resetCardReader (boolean *shouldReset*)

The callback function that must be executed by the app when onReaderResetRequested() is called by the SDK.

Examples: callback.resetCardReader(true); callback.resetCardReader(false);

Parameters

shouldReset	The answer to the question: "Should the card reader be reset?".

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/CardReaderHandler.java

3.6 CardReaderStatus Enum Reference

Public Attributes

- NOT_CONNECTED =(0)

- WAITING_FOR_CARD =(1)
- TOKENIZING =(2)
- STOPPED =(3)
- CONNECTED =(4)
- SWIPE_DETECTED =(5)
- CHECK_CARD_ORIENTATION =(6)
- CHECKING_READER =(7)
- CONFIGURING_READER =(8)
- SHOULD_NOT_SWIPE_EMV_CARD =(9)
- CHIP_ERROR_SWIPE_CARD =(10)
- CARD_DIPPED =(11)
- AUTHORIZING =(12)
- SWIPE_ERROR_SWIPE_AGAIN =(13)

3.6.1 Detailed Description

The Enum CardReaderStatus defines all the statuses that can be returned by the swiper.

3.6.2 Member Data Documentation

3.6.2.1 AUTHORIZING =(12)

Authorizing.

3.6.2.2 CARD_DIPPED =(11)

Card dipped.

3.6.2.3 CHECK_CARD_ORIENTATION =(6)

Check card orientation.

3.6.2.4 CHECKING_READER =(7)

Checking reader.

3.6.2.5 CHIP_ERROR_SWIPE_CARD =(10)

Chip error, swipe card.

3.6.2.6 CONFIGURING_READER =(8)

Configuring reader.

3.6.2.7 CONNECTED =(4)

Connected.

3.6.2.8 NOT_CONNECTED =(0)

Not connected.

3.6.2.9 SHOULD_NOT_SWIPE_EMV_CARD =(9)

Should not swipe EMV card.

3.6.2.10 STOPPED =(3)

Stopped.

3.6.2.11 SWIPE_DETECTED =(5)

Swipe detected.

3.6.2.12 SWIPE_ERROR_SWIPE_AGAIN =(13)

Swipe error, swipe again.

3.6.2.13 TOKENIZING =(2)

Tokenizing.

3.6.2.14 WAITING_FOR_CARD =(1)

Waiting for card.

The documentation for this enum was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/enums/CardReaderStatus.java

3.7 CardReaderHandler.CardReaderTransactionInfoCallback Interface Reference

Public Member Functions

- void useTransactionInfo (double amount, CurrencyCode currencyCode, long accountId)

3.7.1 Detailed Description

The Interface CardReaderTransactionInfoCallback defines the method used to provide transaction information to the card reader before a transaction.

3.7.2 Member Function Documentation

3.7.2.1 void useTransactionInfo (double *amount,* CurrencyCode *currencyCode,* long *accountId*)

The callback function that must be executed when onTransactionInfoRequested() is called by the SDK. Note: In the staging environment, use amounts of 20.61, 120.61, 23.61 and 123.61 to simulate authorization errors. Amounts of 21.61, 121.61, 22.61 and 122.61 will simulate successful auth.

Example: callback.useTransactionInfo(21.61, CurrencyCode.USD, 1234567);

Parameters

amount	the amount for the transaction. It will be rounded to the nearest two decimal places.
currencyCode	the currency code for the transaction. e.g. CurrencyCode.USD.
accountId	the WePay account id of the merchant.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/CardReaderHandler.java

3.8 CheckoutHandler Interface Reference

Public Member Functions

- void onSuccess (String signatureUrl, String checkoutId)
- void onError (Bitmap image, String checkoutId, Error error)

3.8.1 Detailed Description

The Interface CheckoutHandler defines the methods used to return results of a storeSignature operation.

3.8.2 Member Function Documentation

3.8.2.1 void onError (Bitmap *image*, String *checkoutId*, Error *error*)

Gets called when an error occurs while storing a signature.

Parameters

image	the signature image to be stored.
checkoutId	the checkout id associated with the signature.
error	the error which caused the failure.

3.8.2.2 void onSuccess (String *signatureUrl*, String *checkoutId*)

Gets called when a signature is successfully stored for the given checkout id.

Parameters

signatureUrl	the url for the signature image.
checkoutId	the checkout id associated with the signature.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/CheckoutHandler.java

3.9 Config Class Reference

Public Member Functions

- Config (Context context, String clientId, String environment)
- Context getContext ()
- String getClientId ()
- String getEnvironment ()
- boolean isUseLocation ()
- Config setUseLocation (boolean useLocation)
- boolean isUseTestEMVCards ()
- Config setUseTestEMVCards (boolean useTestEMVCards)
- boolean shouldRestartCardReaderAfterSuccess ()
- Config setRestartCardReaderAfterSuccess (boolean restartCardReaderAfterSuccess)
- boolean shouldRestartCardReaderAfterGeneralError ()
- Config setRestartCardReaderAfterGeneralError (boolean restartCardReaderAfterGeneralError)
- boolean shouldRestartCardReaderAfterOtherErrors ()
- Config setRestartCardReaderAfterOtherErrors (boolean restartCardReaderAfterOtherErrors)

Static Public Attributes

- final static String ENVIRONMENT_STAGE = "stage"
- final static String ENVIRONMENT_PRODUCTION = "production"

3.9.1 Detailed Description

The Class Config contains the configuration required to initialize the sdk.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Config (Context *context,* String *clientId,* String *environment*)

Instantiates a new config.

Parameters

context	the application context
clientId	the client id for your WePay app
environment	the environment (use one of the provided constants - ENVIRONMENT_STAGING or ENVIRONMENT_PRODUCTION)

3.9.3 Member Function Documentation

3.9.3.1 String getClientId ( )

Gets the client id.

Returns

the client id

3.9.3.2 Context getContext ( )

Gets the context.

Returns

the context

3.9.3.3 String getEnvironment ( )

Gets the environment.

Returns

the environment

3.9.3.4 boolean isUseLocation ( )

Determines if we should use location services.

Returns

the use location config

3.9.3.5 boolean isUseTestEMVCards ( )

Determines if we should use test EMV cards.

Returns

the use test EMV cards config

3.9.3.6 Config setRestartCardReaderAfterGeneralError (boolean *restartCardReaderAfterGeneralError*)

Sets the option for the card reader to automatically restart after a general error (errorCategory:ERROR_CATEGORY↩
_CARD_READER, errorCode:CARD_READER_GENERAL_ERROR). If not explicitly set to false, defaults to true.

Parameters

restartCardReaderAfterGeneralError	the flag to determine if the card reader should automatically restart after a general error.

Returns

the config

3.9.3.7 Config setRestartCardReaderAfterOtherErrors (boolean *restartCardReaderAfterOtherErrors*)

Sets the option for the card reader to automatically restart after an error other than general error. If not explicitly set to true, defaults to false.

Parameters

restartCardReaderAfterOtherErrors	the flag to determine if the card reader should automatically restart after an error other than general error.

Returns

the config

3.9.3.8 Config setRestartCardReaderAfterSuccess (boolean *restartCardReaderAfterSuccess*)

Sets the option for the card reader to automatically restart after a successful swipe. If not explicitly set to true, defaults to false.

Parameters

restartCardReaderAfterSuccess	the flag to determine if the card reader should automatically restart after a successful swipe.

Returns

the config

3.9.3.9 Config setUseLocation (boolean *useLocation*)

Sets the option for using location services for fraud detection purposes. If not explicitly set to true, defaults to false.

Parameters

useLocation	the permission to use location

Returns

the config

3.9.3.10 Config setUseTestEMVCards (boolean *useTestEMVCards*)

Sets the option for using test EMV cards. If not explicitly set to true, defaults to false.

Parameters

useTestEMVCards	the permission to use location

Returns

the config

3.9.3.11 boolean shouldRestartCardReaderAfterGeneralError ( )

Determines if the card reader should automatically restart after a general error (errorCategory:ERROR_CATEGORY↩
_CARD_READER, errorCode:CARD_READER_GENERAL_ERROR).

Returns

true, if the card reader restarts after a general error

3.9.3.12 boolean shouldRestartCardReaderAfterOtherErrors ( )

Determines if the card reader should automatically restart after an error other than general error.

Returns

true, if the card reader restarts after an error other than general error.

3.9.3.13 boolean shouldRestartCardReaderAfterSuccess ( )

Determines if the card reader should automatically restart after a successful swipe.

Returns

true, if the card reader restarts after success

3.9.4 Member Data Documentation

3.9.4.1 final static String ENVIRONMENT_PRODUCTION = "production" [static]

The constant string representing the production environment.

3.9.4.2 final static String ENVIRONMENT_STAGE = "stage" [static]

The constant string representing the staging environment.

The documentation for this class was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/models/Config.↩ java

3.10 CurrencyCode Enum Reference

Public Attributes

- USD =(0)

3.10.1 Detailed Description

The Enum CurrencyCode defines all currency codes supported by the sdk.

3.10.2 Member Data Documentation

3.10.2.1 USD =(0)

USD

The documentation for this enum was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/enums/Currency↩ Code.java

3.11 Error Class Reference

Inherits Exception.

Public Member Functions

- String getErrorCategory ()
- String getErrorDomain ()
- String getErrorDescription ()
- Integer getErrorCode ()
- Exception getInnerException ()

Static Public Attributes

- final static String ERROR_DOMAIN_API = "com.wepay.api"
- final static String ERROR_DOMAIN_SDK = "com.wepay.sdk"
- final static String ERROR_CATEGORY_CARD_READER = "card_reader_error"
- final static String ERROR_CATEGORY_API = "api_error"
- final static String ERROR_CATEGORY_SDK = "sdk_error"

3.11.1 Detailed Description

The Class Error contains information about an error that occurs in the sdk.

3.11.2 Member Function Documentation

3.11.2.1 String getErrorCategory ( )

Gets the error category.

Returns

the error category

3.11.2.2 Integer getErrorCode ( )

Gets the error code.

Returns

the error code

3.11.2.3 String getErrorDescription ( )

Gets the error description.

Returns

the error description

3.11.2.4 String getErrorDomain ( )

Gets the error domain.

Returns

the error domain

3.11.2.5 Exception getInnerException ( )

Gets the inner exception.

Returns

the inner exception

3.11.3 Member Data Documentation

3.11.3.1 final static String ERROR_CATEGORY_API = "api_error" [static]

The constant string representing the error category API Error.

3.11.3.2 final static String ERROR_CATEGORY_CARD_READER = "card_reader_error" [static]

The constant string representing the error category Card reader Error.

3.11.3.3 final static String ERROR_CATEGORY_SDK = "sdk_error" [static]

The constant string representing the error category SDK Error.

3.11.3.4 final static String ERROR_DOMAIN_API = "com.wepay.api" [static]

The constant ERROR_DOMAIN_API

3.11.3.5 final static String ERROR_DOMAIN_SDK = "com.wepay.sdk" [static]

The constant ERROR_DOMAIN_SDK

The documentation for this class was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/models/Error.↩ java

3.12 ErrorCode Enum Reference

Public Attributes

- UNKNOWN_ERROR =(10000)
- NO_DATA_RETURNED_ERROR =(10015)
- CARD_READER_GENERAL_ERROR =(10016)
- CARD_READER_INITIALIZATION_ERROR =(10017)
- CARD_READER_TIME_OUT_ERROR =(10018)
- CARD_READER_STATUS_ERROR =(10019)
- INVALID_SIGNATURE_IMAGE_ERROR =(10020)
- NAME_NOT_FOUND_ERROR =(10021)
- INVALID_CARD_DATA =(10022)
- CARD_NOT_SUPPORTED =(10023)
- EMV_TRANSACTION_ERROR =(10024)
- INVALID_APPLICATION_ID =(10025)
- DECLINED_BY_CARD =(10026)
- CARD_BLOCKED =(10027)
- CARD_DECLINED_BY_ISSUER =(10028)
- ISSUER_UNREACHABLE =(10029)
- INVALID_TRANSACTION_INFO =(10030)
- TRANSACTION_INFO_NOT_PROVIDED =(10031)
- PAYMENT_METHOD_CANNOT_BE_TOKENIZED =(10032)

3.12.1 Detailed Description

The Enum ErrorCode defines all error codes returned by the sdk itself. For error codes returned by the server api, visit https://www.wepay.com/developer/reference/errors

3.12.2 Member Data Documentation

3.12.2.1 CARD_BLOCKED =(10027)

The card blocked error.

3.12.2.2 CARD_DECLINED_BY_ISSUER =(10028)

The declined by issuer error.

3.12.2.3 CARD_NOT_SUPPORTED =(10023)

The card not supported error.

3.12.2.4 CARD_READER_GENERAL_ERROR =(10016)

The card reader general error.

3.12.2.5 CARD_READER_INITIALIZATION_ERROR =(10017)

The card reader initialization error.

3.12.2.6 CARD_READER_STATUS_ERROR =(10019)

The card reader status error.

3.12.2.7 CARD_READER_TIME_OUT_ERROR =(10018)

The card reader time out error.

3.12.2.8 DECLINED_BY_CARD =(10026)

The declined by card error.

3.12.2.9 EMV_TRANSACTION_ERROR =(10024)

The EMV transaction error.

3.12.2.10 INVALID_APPLICATION_ID =(10025)

The invalid application error.

3.12.2.11 INVALID_CARD_DATA =(10022)

The invalid card data error.

3.12.2.12 INVALID_SIGNATURE_IMAGE_ERROR =(10020)

The invalid signature image error.

3.12.2.13 INVALID_TRANSACTION_INFO =(10030)

The invalid transaction info.

3.12.2.14 ISSUER_UNREACHABLE =(10029)

The issuer unreachable error.

3.12.2.15 NAME_NOT_FOUND_ERROR =(10021)

The name not found error.

3.12.2.16 NO_DATA_RETURNED_ERROR =(10015)

The no data returned error.

3.12.2.17 PAYMENT_METHOD_CANNOT_BE_TOKENIZED =(10032)

The payment method cannot be tokenized error.

3.12.2.18 TRANSACTION_INFO_NOT_PROVIDED =(10031)

The transaction info not provided error.

3.12.2.19 UNKNOWN_ERROR =(10000)

The unknown error.

The documentation for this enum was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/enums/Error↩ Code.java

3.13 PaymentInfo Class Reference

Public Member Functions

- PaymentInfo (String firstName, String lastName, String email, String paymentDescription, Address billingAddress, Address shippingAddress, PaymentMethod paymentMethod, String ccNumber, String cvv, String expMonth, String expYear, boolean virtualTerminal)
- String getFirstName ()
- String getLastName ()
- String getEmail ()
- String getPaymentDescription ()
- Address getBillingAddress ()
- Address getShippingAddress ()
- PaymentMethod getPaymentMethod ()
- Object getManualInfo ()
- boolean isVirtualTerminal ()
- void addEmail (String email)
- String getFullName ()

3.13.1 Detailed Description

The Class PaymentInfo represents all the information obtained via a particular payment method.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 PaymentInfo (String *firstName,* String *lastName,* String *email,* String *paymentDescription,* Address *billingAddress,* Address *shippingAddress,* PaymentMethod *paymentMethod,* String *ccNumber,* String *cvv,* String *expMonth,* String *expYear,* boolean *virtualTerminal*)

Instantiates a new payment info. Use this constructor when representing manually obtained card data. Note: For virtual terminal, name is optional. A placeholder name will be inserted if it is not provided.

Parameters

firstName	the first name
lastName	the last name
email	the email
paymentDescription	the payment description
billingAddress	the billing address
shippingAddress	the shipping address
paymentMethod	the payment method
ccNumber	the cc number
cvv	the cvv
expMonth	the expiration month
expYear	the expiration year
virtualTerminal	the virtual terminal flag

3.13.3 Member Function Documentation

3.13.3.1 void addEmail (String *email*)

Allows adding an email if one is not already present. The call will be ignored if an email is already present.

Parameters

email	the email to be added

3.13.3.2 Address getBillingAddress ( )

Gets the billing address.

Returns

the billingAddress

3.13.3.3 String getEmail ( )

Gets the email.

Returns

the email

3.13.3.4 String getFirstName ( )

Gets the first name.

Returns

the firstName

3.13.3.5 String getFullName ( )

Gets the full name

Returns

full name if available, otherwise null

3.13.3.6 String getLastName ( )

Gets the last name.

Returns

the lastName

3.13.3.7 Object getManualInfo ( )

Gets the manual info.

Returns

the manualInfo

3.13.3.8 String getPaymentDescription ( )

Gets the payment description.

Returns

the paymentDescription

3.13.3.9 PaymentMethod getPaymentMethod ( )

Gets the payment method.

Returns

the paymentMethod

3.13.3.10 Address getShippingAddress ( )

Gets the shipping address.

Returns

the shippingAddress

3.13.3.11 boolean isVirtualTerminal ( )

Determines if the card info was obtained via Virtual Terminal.

Returns

true if virtual terminal, else false

The documentation for this class was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/models/PaymentInfo.java

3.14 PaymentMethod Enum Reference

Public Attributes

- MANUAL =(0)
- SWIPE =(1)
- DIP =(2)

3.14.1 Detailed Description

The Enum PaymentMethod defines all the payment methods available in the sdk.

3.14.2 Member Data Documentation

3.14.2.1 DIP =(2)

Dip

3.14.2.2 MANUAL =(0)

Manual.

3.14.2.3 SWIPE =(1)

Swipe.

The documentation for this enum was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/enums/Payment↩ Method.java

3.15 PaymentToken Class Reference

Public Member Functions

- PaymentToken (String tokenId)
- String getTokenId ()

3.15.1 Detailed Description

The Class PaymentToken represents payment information that was obtained from the user and is stored on WePay servers. This token can be used to complete the payment transaction via WePay's web APIs.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 PaymentToken (String *tokenId*)

Instantiates a new payment token.

Parameters

tokenId	the token id

3.15.3 Member Function Documentation

3.15.3.1 String getTokenId ( )

Gets the token id.

Returns

the token id

The documentation for this class was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/models/PaymentToken.java

3.16 TokenizationHandler Interface Reference

Public Member Functions

- void onSuccess (PaymentInfo paymentInfo, PaymentToken token)
- void onError (PaymentInfo paymentInfo, Error error)

3.16.1 Detailed Description

The Interface TokenizationHandler defines the method used to return data in response to a tokenization call.

3.16.2 Member Function Documentation

3.16.2.1 void onError (PaymentInfo *paymentInfo,* Error *error*)

Gets called when a tokenization call fails.

Parameters

paymentInfo	the payment info.
error	the error due to which tokenization failed.

3.16.2.2 void onSuccess (PaymentInfo *paymentInfo,* PaymentToken *token*)

Gets called when a tokenization calls succeeds.

Parameters

paymentInfo	the payment info passed to the tokenization call.
token	the token representing the payment info.

The documentation for this interface was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/TokenizationHandler.java

3.17 WePay Class Reference

Public Member Functions

- WePay (Config config)
- void startCardReaderForReading (CardReaderHandler cardReaderHandler)
- void startCardReaderForTokenizing (CardReaderHandler cardReaderHandler, TokenizationHandler tokenizationHandler, AuthorizationHandler authorizationHandler)
- void stopCardReader ()
- void tokenize (final PaymentInfo paymentInfo, final TokenizationHandler tokenizationHandler)
- void storeSignatureImage (final Bitmap image, final String checkoutId, final CheckoutHandler checkoutHandler)

3.17.1 Detailed Description

Main Class containing all public endpoints.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 WePay (Config *config*)

Instantiates a new WePay instance.

Parameters

config	the WePay config

3.17.3 Member Function Documentation

3.17.3.1 void startCardReaderForReading (CardReaderHandler *cardReaderHandler*)

Use this method if you just want to read non-sensitive data from the card, without actually charging the card. Non-sensitive info from the card will be returned via the CardReaderHandler interface.

The reader will wait 60 seconds for a card, and then return a timout error if a card is not detected. The reader will automatically stop waiting for card if:

- a timeout occurs
- a successful swipe/dip is detected
- an unexpected error occurs
- stopReader is called

However, if a general error (errorCategory:ERROR_CATEGORY_CARD_READER, errorCode:CARD_READER_GENERAL_ERROR) occurs while reading, after a few seconds delay, the reader will automatically start waiting again for another 60 seconds. At that time, CardReaderHandler's onStatusChange() method will be called with status = WAITING_FOR_CARD, and the user can try to swipe/dip again. This behavior can be configured with com.wepay.android.models.Config.

WARNING: When this method is called, a (normally inaudible) signal is sent to the headphone jack of the phone, where the reader is expected to be connected. If headphones are connected instead of the reader, they may emit a very loud audible tone on receiving this signal. This method should only be called when the user intends to use the reader.

Parameters

cardReaderHandler	the card reader handler

3.17.3.2 void startCardReaderForTokenizing (CardReaderHandler *cardReaderHandler,* TokenizationHandler *tokenizationHandler,* AuthorizationHandler *authorizationHandler*)

Use this method if you want to tokenize the card info. Non-sensitive info from the card will be returned via the CardReaderHandler interface. The card info will be tokenized by WePay's servers, and the token will be returned via the TokenizationHandler interface.

The reader will wait 60 seconds for a card, and then return a timout error if a card is not detected. The reader will automatically stop waiting for card if:

- a timeout occurs
- a successful swipe/dip is detected
- an unexpected error occurs
- stopReader is called

However, if a general error (errorCategory:ERROR_CATEGORY_CARD_READER, errorCode:CARD_READER_GENERAL_ERROR) occurs while reading, after a few seconds delay, the reader will automatically start waiting again for another 60 seconds. At that time, CardReaderHandler's onStatusChange() method will be called with status =

WAITING_FOR_CARD, and the user can try to swipe/dip again. This behavior can be configured with com.wepay.↩ android.models.Config.

WARNING: When this method is called, a (normally inaudible) signal is sent to the headphone jack of the phone, where the reader is expected to be connected. If headphones are connected instead of the reader, they may emit a very loud audible tone on receiving this signal. This method should only be called when the user intends to use the reader.

Parameters

cardReaderHandler	the card reader handler
tokenizationHandler	the tokenization handler
authorizationHandler	the authorization handler

3.17.3.3 void stopCardReader ( )

Stops the reader. In response, CardReaderHandler's onStatusChange() method will be called with status = STOPPED. Any tokenization in progress will not be stopped, and its result will be delivered to the TokenizationHandler.

3.17.3.4 void storeSignatureImage (final Bitmap *image,* final String *checkoutId,* final **CheckoutHandler** *checkoutHandler*)

Use this method to store a signature image associated with a checkout id on WePay's servers. The signature can be retrieved via a server-to-server call that fetches the checkout object. The aspect ratio (width:height) of the image must be between 1:4 and 4:1. If needed, the image will internally be scaled to fit inside 256x256 pixels, while maintaining the original aspect ratio.

Parameters

image	the signature image to be stored.
checkoutId	the checkout id associated with the signature
checkoutHandler	the signature handler

3.17.3.5 void tokenize (final **PaymentInfo** *paymentInfo,* final **TokenizationHandler** *tokenizationHandler*)

Use this method to tokenize any PaymentInfo object, such as one representing credit card info obtained manually. The payment info will be tokenized by WePay's servers, and the token will be returned via the TokenizationHandler interface.

Parameters

paymentInfo	the payment info to be tokenized
tokenizationHandler	the tokenization handler

The documentation for this class was generated from the following file:

- /Users/chaitanya.bagaria/sandbox/wepay-android-priv/WePay/app/src/main/java/com/wepay/android/WePay.java

Index