WePay iOS SDK

7.0.0

Generated by Doxygen 1.8.12

Contents

1 Getting Started

Introduction

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

It is meant for consumption by `WePay` partners who are developing their own iOS 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 iOS SDK supports the following payment methods

- 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 stripe 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

Using `cocoapods` (recommended)

- Add `pod "WePay"` to your podfile
- Run `pod install`
- Done!

The `SwiftExample app` also utilizes `cocoapods`.

Using library binaries

- Download the latest zip file from our `releases page`
- Unzip the file and copy the contents anywhere inside your project directory
- In Xcode, go to your app's target settings. On the `Build Phases` tab, expand the `Link Binary With Libraries` section.
- Include the following iOS frameworks:
  - AudioToolbox.framework
  - AVFoundation.framework
  - CoreBluetooth.framework
  - CoreLocation.framework
  - CoreTelephony.framework
  - ExternalAccessory.framework
  - MediaPlayer.framework
  - SystemConfiguration.framework
  - UIKit.framework
  - libstdc++.6.0.9.dylib
  - libstdc++.dylib
  - libz.dylib
- Also include the framework files you copied:
  - WePay.framework
- Done!

Note: Card reader functionality is not available in this SDK by default. If you are interested in using the WePay Card Reader, please contact your sales representative or account manager. If you have yet to be in direct contact with WePay, please email `sales@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`.

General documentation about the WePay mobile point of sale (mPOS) program is available `here`.

SDK Organization

WePay.h

`WePay.h` is the starting point for consuming the SDK, and contains 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.

Delegate protocols

The SDK uses delegate protocols to respond to API calls. You must adopt the relevant protocols to receive responses to the API calls you make. Detailed reference documentation is available on the reference page for each protocol:

- `WPAuthorizationDelegate`
- `WPBatteryLevelDelegate`
- `WPCardReaderDelegate`
- `WPCheckoutDelegate`
- `WPTokenizationDelegate`

Data Models

All other classes in the SDK are data models 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 `documentation` 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

`WPError.h` serves as documentation for all errors surfaced by the WePay iOS SDK.

Samples

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

See the `SwiftExample app` for a working implementation of all API methods in a Swift 3 application. Note: make sure to open the project using `SwiftApp.xcworkspace` and not `SwiftApp.xcodeproj`.

Initializing a Bridging Header (for Swift apps)

- For using Objective-C modules in a Swift application, you will need to create a bridging header.
- Make sure you are working in `{app_name}.xcworkspace` file.
- Under your target application folder, create a header file: `{app_name}-Bridging-Header.h`
- In the Header file, import the modules you need:

```
#import <WePay/WePay.h>
```

- Click on the main application project to get to `Build Settings`.
- Search for `bridging header` in your target application to find a setting called `Swift Compiler - Code Generation`.
- Double click in the column next to `Objective-C Bridging Header` and add your Header file: `{app_name}/{app_name}-Bridging-Header.h`
- There's no need to import the module in your code; you can use the module by calling it directly in your Swift application.

Initializing the SDK

- Complete the installation steps (above).
- Include WePay.h

```
#import <WePay/WePay.h>
```

- Define a property to store the WePay object

```
\@property (nonatomic, strong) WePay *wepay;
```

- Create a WPConfig object

```
WPConfig *config = [[WPConfig alloc] initWithClientId:@"your_client_id" environment:
      kWPEnvironmentStage];
```

- Initialize the WePay object and assign it to the property

```
self.wepay = [[WePay alloc] initWithConfig:config];
```

Providing permission to use microphone for card reader communication

- Open your app's Info.plist file and add an entry for NSMicrophoneUsageDescription.

```
<key>NSMicrophoneUsageDescription</key>
<string>Microphone permission is required for operating card reader</string>
```

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

- In Xcode, go to your projects settings. On the Build Phases tab, expand the Link Binary With Libraries section and include the CoreLocation.framework iOS framework.
- Open your app's Info.plist file and add entries for NSLocationUsageDescription and NSLocationWhenInUse↩ UsageDescription.

```
<key>NSLocationUsageDescription</key>
<string>Location information is used for fraud prevention</string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>Location information is used for fraud prevention</string>
```

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

```
config.useLocation = YES;
```

Integrating the Card Reader payment methods (Swipe+Dip)

- Adopt the WPCardReaderDelegate, WPTokenizationDelegate, and WPAuthorizationDelegate protocols

```
\@interface MyWePayDelegate : NSObject <WPCardReaderDelegate,
      WPTokenizationDelegate, WPAuthorizationDelegate>
```

- Implement the WPCardReaderDelegate protocol methods

```
- (void) cardReaderDidChangeStatus:(id) status
{
    if (status == kWPCardReaderStatusNotConnected) {
        // show UI that prompts the user to connect the Card Reader
        self.statusLabel.text = @"Connect Card Reader";
    } else if (status == kWPCardReaderStatusWaitingForSwipe) {
        // show UI that prompts the user to swipe
        self.statusLabel.text = @"Swipe Card";
    } else if (status == kWPCardReaderStatusSwipeDetected) {
        // provide feedback to the user that a swipe was detected
        self.statusLabel.text = @"Swipe Detected...";
    } else if (status == kWPCardReaderStatusTokenizing) {
        // provide feedback to the user that the card info is being tokenized/verified
        self.statusLabel.text = @"Tokenizing...";
    }  else if (status == kWPCardReaderStatusStopped) {
        // provide feedback to the user that the swiper has stopped
        self.statusLabel.text = @"Card Reader Stopped";
    } else {
        // handle any other status messages
        self.statusLabel.text = [status description];
    }
}

- (void) selectCardReader:(NSArray *)cardReaderNames
               completion:(void (^)(NSInteger selectedIndex))completion
{
    // In production apps, the merchant must choose the card reader they want to use.
    // Here, we always select the first card reader in the array
    int selectedIndex = 0;
    completion(selectedIndex);
}

- (void) shouldResetCardReaderWithCompletion:(void (^)(BOOL))completion
{
    // Change this to YES if you want to reset the card reader
    completion(NO);
}

- (void) authorizeAmountWithCompletion:(void (^)(NSDecimalNumber *amount, NSString *currencyCode, long
      accountId))completion
{
    // obtain transaction info
    double amount = @(10.00);
    NSString *currencyCode = @"USD";
    long accountId = 12345678;
    // execute the completion
    completion(amount, currencyCode, accountId);
}

- (void) selectEMVApplication:(NSArray *)applications
                   completion:(void (^)(NSInteger selectedIndex))completion
{
    // In production apps, the payer must choose the app id they want to use.
    // Here, we always select the first application in the array
    int selectedIndex = 0;
    completion(selectedIndex);
}

- (void) insertPayerEmailWithCompletion:(void (^)(NSString *email))completion
{
    // obtain email
    NSString *email = @"emv@example.com";

    // execute the completion
    completion(email);
}

- (void) didReadPaymentInfo:(WPPaymentInfo *)paymentInfo
{
    // use the payment info (for display/recordkeeping)
    // wait for tokenization(swipe)/authorization(dip) response
}

- (void) didFailToReadPaymentInfoWithError:(NSError *)error
{
    // Handle the error
}
```

- Implement the WPTokenizationDelegate protocol methods

```
- (void) paymentInfo:(WPPaymentInfo *)paymentInfo didTokenize:(
      WPPaymentToken *)paymentToken
{
    // Send the tokenId (paymentToken.tokenId) to your server
    // Your server would use the tokenId to make a /checkout/create call to complete the transaction
}

- (void) paymentInfo:(WPPaymentInfo *)paymentInfo didFailTokenization:(NSError *)error
{
    // Handle the error
}
```

- Implement the WPAuthorizationDelegate protocol methods

```
- (void) paymentInfo:(WPPaymentInfo *)paymentInfo
        didAuthorize:(WPAuthorizationInfo *)authorizationInfo
{
    // Send the token Id (authorizationInfo.tokenId) and transaction token
       (authorizationInfo.transactionToken) to your server
    // Your server must use the tokenId and transactionToken to make a /checkout/create call to complete
       the transaction
}

- (void) paymentInfo:(WPPaymentInfo *)paymentInfo
didFailAuthorization:(NSError *)error
{
    // Handle the error
}
```

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

```
[self.wepay startCardReaderForTokenizingWithCardReaderDelegate:self tokenizationDelegate:self
      authorizationDelegate:self];
// 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), or powers on their bluetooth card reader.

2. The SDK tries to detect the card reader and initialize it.
   - The `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusSearching`.
   - If any card readers are discovered, the `selectCardReader:` method will be called with an array of discovered devices. If anything is plugged into the headphone jack, `"AUDIOJACK"` will be one of the devices discovered.
   - If no card readers are detected, the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusNotConnected`.
   - Once the card reader selection completion block is called, the SDK will attempt to to connect to the selected card reader.
   - If the card reader is successfully connected, then the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusConnected`.
3. Next, the SDK checks if the card reader is correctly configured (the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusCheckingReader`).
   - If the card reader is already configured, the app is given a chance to force configuration. The SDK calls the `shouldResetCardReaderWithCompletion:` method, and the app must execute the completion method, telling the SDK whether or not the reader should be reset.
   - If the reader was not already configured, or the app requested a reset, the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusConfiguringReader` and the card reader is configured.
4. Next, if the card reader is successfully initialized, the SDK asks the app for transaction information by calling the `authorizeAmountWithCompletion:` method. The app must execute the completion method, telling the SDK what the amount, currency code and merchant account id is.
5. Next, the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusWaitingForCard`.
6. If the user swipes a card successfully:
   - The `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusSwipeDetected`.
   - The SDK attempts to ask the app for the payer's email by calling the `insertPayerEmailWithCompletion:` method. If the app implements this optional delegate method, it must execute the completion method and pass in the payer's email address.
   - The `didReadPaymentInfo:` method is called with the obtained payment info.
   - The `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusTokenizing`, and the SDK will automatically send the obtained card info to WePay's servers for tokenization.
   - If tokenization succeeds, the `paymentInfo:didTokenize:` method will be called.
   - If tokenization fails, the `paymentInfo:didFailTokenization:` method will be called with the appropriate error, and processing will stop.
7. Instead, if the user dips a card successfully:
   - The `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusCardDipped`
   - If the card has multiple applications on it, the payer must choose one:
     - The SDK calls the `selectEMVApplication:completion:` 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 completion method and passing in the index of the chosen application.
  - Next, the SDK obtains card data from the chip on the card.
  - The SDK attempts to ask the app for the payer's email by calling the `insertPayerEmailWith↩ Completion:` method. If the app implements this optional delegate method, it must execute the completion method and pass in the payer's email address.
  - The `didReadPaymentInfo:` method is called with the obtained payment info.
  - The `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReader↩ StatusAuthorizing`, and the SDK will automatically send the obtained EMV card info to WePay's servers for authorization.
  - If authorization succeeds, the `paymentInfo:didAuthorize:` method will be called and processing will stop.
  - If authorization fails, the `paymentInfo:didFailAuthorization:` method will be called.

8. If a recoverable error occurs during swiping or dipping, one of the failure methods will be called. After a few seconds, the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReader↩ StatusWaitingForCard` and the card reader will again wait for the user to swipe/dip a card.

9. If an unrecoverable error occurs, or if the SDK is unable to obtain data from the card, one of teh failure methods will be called with the appropriate error.

10. When processing stops, the `cardReaderDidChangeStatus:` method will be called with `status = k↩ WPCardReaderStatusStopped`.

1. Done!

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

Integrating the Manual payment method

- Adopt the WPTokenizationDelegate protocol

```
\@interface MyWePayDelegate : NSObject <WPTokenizationDelegate>
```

- Implement the WPTokenizationDelegate protocol methods

```
- (void) paymentInfo:(WPPaymentInfo *)paymentInfo didTokenize:(
      WPPaymentToken *)paymentToken
{
    // Send the tokenId (paymentToken.tokenId) to your server
    // Your server can use the tokenId to make a /checkout/create call to complete the transaction
}

- (void) paymentInfo:(WPPaymentInfo *)paymentInfo didFailTokenization:(NSError *)error
{
    // Handle error
}
```

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

```
WPPaymentInfo *paymentInfo = [[WPPaymentInfo alloc] initWithFirstName:@"WPiOS"
                                                         lastName:@"Example"
                                                            email:@"wp.ios.example@wepay.com"
                                                   billingAddress:[[
      WPAddress alloc] initWithZip:@"94306"]
                                                  shippingAddress:nil
                                                       cardNumber:@"5496198584584769"
                                                              cvv:@"123"
                                                         expMonth:@"04"
                                                          expYear:@"2020"
                                                  virtualTerminal:YES];
// Note: the virtualTerminal parameter above should be set to YES if a merchant is collecting payments
       manually using your app. It should be set to NO if a payer is making a manual payment using your app.
```

- Make the WePay API call, passing in the instance of the class that implemented the WPTokenizationDelegate protocol

```
[self.wepay tokenizeManualPaymentInfo:paymentInfo tokenizationDelegate:self];
```

- That's 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, the `paymentInfo:didTokenize:` method will be called
3. Otherwise, if the tokenization fails, the `paymentInfo:didFailTokenization:` method will be called with the appropriate error

Integrating the Store Signature API

- Adopt the WPCheckoutDelegate protocol

```
\@interface MyWePayDelegate : NSObject <WPCheckoutDelegate>
```

- Implement the WPCheckoutDelegate protocol methods

```
- (void) didStoreSignature:(NSString *)signatureUrl
             forCheckoutId:(NSString *)checkoutId
{
    // success! nothing to do here
}

- (void) didFailToStoreSignatureImage:(UIImage *)image
                        forCheckoutId:(NSString *)checkoutId
                            withError:(NSError *)error
{
    // handle the error
}
```

- Obtain the checkout_id associated with this signature from your server

```
NSString *checkoutId = [self obtainCheckoutId];
```

- Instantiate a UIImage containing the user's signature

```
UIImage *signature = [UIImage imageNamed:@"dd_signature.png"];
```

- Make the WePay API call, passing in the instance of the class that implemented the WPCheckoutDelegate protocol

```
[self.wepay storeSignatureImage:signature
                  forCheckoutId:checkoutId
               checkoutDelegate:self];
```

- That's it! The following sequence of events will occur:
  1. The SDK will send the obtained signature to WePay's servers
  2. If the operation succeeds, the `didStoreSignature:forCheckoutId:` method will be called
  3. Otherwise, if the operation fails, the `didFailToStoreSignatureImage:forCheckoutId↩ :withError:` method will be called with the appropriate error

Integrating the the Battery Level API

- Adopt the WPBatteryLevelDelegate protocol

```
\@interface MyWePayDelegate : NSObject <WPBatteryLevelDelegate>
```

- Implement the WPCheckoutDelegate protocol methods

```
- (void) didGetBatteryLevel:(int)batteryLevel
{
    // success! Show the current level to the user.
}

- (void) didFailToGetBatteryLevelwithError:(NSError *)error
{
    // handle the error
}
```

- Make the WePay API call, passing in the instance(s) of the class(es) that implemented the WPCardReader↩ Delegate and WPBatteryLevelDelegate protocols

```
[self.wepay getCardReaderBatteryLevelWithCardReaderDelegate:self batteryLevelDelegate:self];
```

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

1. The SDK will attempt to read the battery level of the card reader
2. If the operation succeeds, WPBatteryLevelDelegate's `didGetBatteryLevel:` method will be called with the result
3. Otherwise, if the operation fails, WPBatteryLevelDelegate's `didFailToGetBatteryLevelwithError:` method will be called with the appropriate error

Configuring the SDK

The experiences described above can be modified by utilizing the configuration options available on the WPConfig object. Detailed descriptions for each configurable property is available in the documentation for WPConfig.

Test/develop using mock card reader and mock WepayClient

- To use mock card reader implementation instead of using the real reader, instantiate a MockConfig object and pass it to Config:

```
WPMockConfig *mockConfig = [[WPMockConfig alloc] init];
config.mockConfig = mockConfig;
```

- To use mock WepayClient implementation instead of interacting with the real WePay server, set the corresponding option on the mockConfig object:

```
mockConfig.useMockWepayClient = YES;
```

- Other options are also available:

```
mockConfig.mockPaymentMethod = kWPPaymentMethodSwipe; // Payment method to mock; Defaults
       to SWIPE.
mockConfig.cardReadTimeOut = YES; // To mock a card reader timeout; Defaults to NO.
mockConfig.cardReadFailure = YES; // To mock a failure for card reading; Defaults to NO.
mockConfig.cardTokenizationFailure = YES; // To mock a failure for card
       tokenization; Defaults to NO.
mockConfig.EMVAuthFailure = YES; // To mock a failure for EMV authorization; Defaults to NO.
mockConfig.multipleEMVApplication = YES; // To mock multiple EMV applications on card
       to choose from; Defaults to NO.
mockConfig.batteryLevelError = YES; // To mock an error while fetching battery level;
       Defaults to NO.
mockConfig.mockCardReaderIsDetected = NO; // To mock a card reader being available
       for connection; Defaults to YES.
```

Integration tests and unit tests

All the integration tests and unit tests are located in the `/WePayTests/` directory.

From Xcode

From the Tests Navigator tab:

- To run a single test, right-click the test method and select "Test <name>".
- To run all test methods in a class, right-click the class and select "Run <name>".
- To run all tests in a directory, right-click the directory and select "Run <name>".
- To run all tests in the project, use the menu option Product > Test or press (Cmd + U).

From the command line

Go to this repo directory and execute:

```
xcodebuild test -project WePay.xcodeproj -scheme "Release Framework"  -destination 'platform=iOS
       Simulator,name=iPhone 7'
```

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

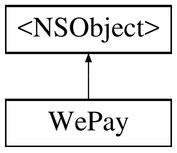
Here is a list of all documented files with brief descriptions:

5 Class Documentation

5.1 WePay Class Reference

```
#import <WePay.h>
```

Inheritance diagram for WePay:



Instance Methods

Initialization

- (instancetype) - initWithConfig:

Tokenization

- (void) - tokenizePaymentInfo:tokenizationDelegate:

Card Reader related methods

- (void) - startTransactionForReadingWithCardReaderDelegate:
- (void) - startTransactionForTokenizingWithCardReaderDelegate:tokenizationDelegate:authorization←Delegate:
- (void) - stopCardReader

Checkout related methods

- (void) - storeSignatureImage:forCheckoutId:checkoutDelegate:

Battery Level related methods

- (void) - getCardReaderBatteryLevelWithCardReaderDelegate:batteryLevelDelegate:

Remember card reader related methods

- (NSString ∗) - getRememberedCardReader
- (void) - forgetRememberedCardReader

Properties

- WPConfig ∗ config

5.1.1 Detailed Description

Main Class containing all public endpoints.

5.1.2 Method Documentation

5.1.2.1 - (void) forgetRememberedCardReader

Clears the name of the most recently used card reader.

5.1.2.2 - (void) getCardReaderBatteryLevelWithCardReaderDelegate: (id< WPCardReaderDelegate >) *cardReaderDelegate* batteryLevelDelegate:(id< WPBatteryLevelDelegate >) *batteryLevelDelegate*

Gets the current battery level of the card reader.

Parameters

cardReaderDelegate	the delegate class which will receive the card reader response(s) for this call.
batteryLevelDelegate	the delegate class which will receive the battery level response(s) for this call.

5.1.2.3 - (NSString ∗) getRememberedCardReader

Gets the name of the most recently used card reader.

Returns

the name of the card reader.

5.1.2.4 - (instancetype) initWithConfig: (WPConfig ∗) *config*

The designated intializer. Use this to initialize the SDK.

Parameters

config	A WPConfig instance.

Returns

A WePay instance, which can be used to access most of the functionality of this sdk.

5.1.2.5 - (void) startTransactionForReadingWithCardReaderDelegate: (id< **WPCardReaderDelegate** >) *cardReaderDelegate*

Initializes the transaction for reading card info.

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

- a timeout occurs
- a card is successfully detected
- an unexpected error occurs
- stopCardReader is called

However, if a general error (domain:kWPErrorCategoryCardReader, errorCode:WPErrorCardReaderGeneralError) occurs while reading, after a few seconds delay, the card reader will automatically start waiting again for another 60 seconds. At that time, WPCardReaderDelegate's cardReaderDidChangeStatus: method will be called with kWPCard↵ReaderStatusWaitingForCard, and the user can try to use the card reader again. This behavior can be configured with WPConfig.

WARNING: When this method is called, if the "AUDIOJACK" device is selected via the onReaderSelection: method in WPCardReaderDelegate, a (normally inaudible) signal is sent to the headphone jack of the phone, where the card reader is expected to be connected. If headphones are connected instead of the card 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 a card reader.

Parameters

cardReaderDelegate	The delegate class which will receive the response(s) for this call.

5.1.2.6 - (void) startTransactionForTokenizingWithCardReaderDelegate: (id< **WPCardReaderDelegate** >) *cardReaderDelegate* tokenizationDelegate:(id< **WPTokenizationDelegate** >) *tokenizationDelegate* authorizationDelegate:(id< **WPAuthorizationDelegate** >) *authorizationDelegate*

Initializes the card reader for reading and then automatically tokenizing card info. If an EMV card is dipped into a connected EMV reader, the card will automatically be authorized.

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

- a timeout occurs
- a card is successfully detected
- an unexpected error occurs
- stopCardReader is called

However, if a general error (domain:kWPErrorCategoryCardReader, errorCode:WPErrorCardReaderGeneralError) occurs while reading, after a few seconds delay, the card reader will automatically start waiting again for another 60 seconds. At that time, WPCardReaderDelegate's cardReaderDidChangeStatus: method will be called with kWPCard←ReaderStatusWaitingForCard, and the user can try to use the card reader again. This behavior can be configured with WPConfig.

WARNING: When this method is called, if the "AUDIOJACK" device is selected via the onReaderSelection: method in WPCardReaderDelegate, a (normally inaudible) signal is sent to the headphone jack of the phone, where the card reader is expected to be connected. If headphones are connected instead of the card 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 a card reader.

Parameters

cardReaderDelegate	The delegate class which will receive the card reader response(s) for this call.
tokenizationDelegate	The delegate class which will receive the tokenization response(s) for this call.
authorizationDelegate	The delegate class which will receive the authorization response(s) for this call.

5.1.2.7 - (void) stopCardReader

Stops the card reader. In response, WPCardReaderDelegate's cardReaderDidChangeStatus: method will be called with kWPCardReaderStatusStopped. The status can only be returned if you've provided a WPCardReaderDelegate by starting a card reader operation after the WePay object was initialized. Any operation in progress may not stop, and its result will be delivered to the appropriate delegate.

5.1.2.8 - (void) storeSignatureImage: (UIImage $*$) *image* forCheckoutId:(NSString $*$) *checkoutId* checkoutDelegate:(id$<$ WPCheckoutDelegate $>$) *checkoutDelegate*

Stores 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 this transaction.
checkoutDelegate	The delegate class which will receive the response(s) for this call.

5.1.2.9 - (void) tokenizePaymentInfo: (WPPaymentInfo $*$) *paymentInfo* tokenizationDelegate:(id$<$ WPTokenizationDelegate $>$) *tokenizationDelegate*

Creates a payment token from a WPPaymentInfo object.

Parameters

paymentInfo	The payment info obtained from the user in any form.
tokenizationDelegate	The delegate class which will receive the tokenization response(s) for this call.

5.1.3 Property Documentation

5.1.3.1 - (WPConfig∗) config `[read], [nonatomic], [strong]`

Your WePay config

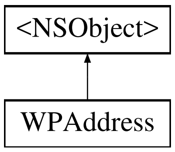
The documentation for this class was generated from the following file:

- WePay.h

5.2 WPAddress Class Reference

```
#import <WPAddress.h>
```

Inheritance diagram for WPAddress:



Instance Methods

- (instancetype) - initWithZip:
- (instancetype) - initWithAddress1:address2:city:state:zip:
- (instancetype) - initWithAddress1:address2:city:region:postcode:country:
- (NSDictionary ∗) - **toDict**

Properties

- NSString ∗ address1
- NSString ∗ address2
- NSString ∗ city
- NSString ∗ country
- NSString ∗ postcode
- NSString ∗ region
- NSString ∗ state
- NSString ∗ zip

5.2.1 Detailed Description

An instance of this class represents a physical address.

5.2.2 Method Documentation

5.2.2.1 - (instancetype) initWithAddress1: (NSString ∗) *address1* address2:(NSString ∗) *address2* city:(NSString ∗) *city* region:(NSString ∗) *region* postcode:(NSString ∗) *postcode* country:(NSString ∗) *country*

Initializes a non-US Address.

Parameters

address1	The first line of the street address.
address2	The second line of the street address.
city	The city.
region	The region. Only for non-US addresses when available.
postcode	The postcode. Only for non-US addresses when available.
country	The 2-letters ISO-3166-1 country code.

Returns

The address.

5.2.2.2 - (instancetype) initWithAddress1: (NSString ∗) *address1* address2:(NSString ∗) *address2* city:(NSString ∗) *city* state:(NSString ∗) *state* zip:(NSString ∗) *zip*

Initializes a US Address.

Parameters

address1	The first line of the street address.
address2	The second line of the street address.
city	The city.
state	The 2-letters US state code.
zip	The US zip or zip-plus code.

Returns

The address.

5.2.2.3 - (instancetype) initWithZip: (NSString ∗) *zip*

Initializes a US Address with just a zip.

Parameters

zip	The US zip or zip-plus code.

Returns

The address.

5.2.3 Property Documentation

5.2.3.1 - (NSString∗) address1 `[read], [nonatomic], [strong]`

The first line of the street address.

5.2.3.2 - (NSString∗) address2 `[read], [nonatomic], [strong]`

The second line of the street address.

5.2.3.3 - (NSString∗) city `[read], [nonatomic], [strong]`

The city.

5.2.3.4 - (NSString∗) country `[read], [nonatomic], [strong]`

The 2-letters ISO-3166-1 country code.

5.2.3.5 - (NSString∗) postcode `[read], [nonatomic], [strong]`

The postcode. Only for non-US addresses when available.

5.2.3.6 - (NSString∗) region `[read], [nonatomic], [strong]`

The region. Only for non-US addresses when available.

5.2.3.7 - (NSString∗) state `[read], [nonatomic], [strong]`

The 2-letters US state code. Only for US addresses.

5.2.3.8 - (NSString∗) zip `[read], [nonatomic], [strong]`

The US zip or zip-plus code. Only for US addresses.

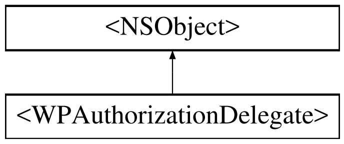
The documentation for this class was generated from the following file:

- WPAddress.h

5.3 <WPAuthorizationDelegate> Protocol Reference

```
#import <WePay.h>
```

Inheritance diagram for <WPAuthorizationDelegate>:



Instance Methods

- (void) - paymentInfo:didAuthorize:
- (void) - paymentInfo:didFailAuthorization:

5.3.1 Detailed Description

This delegate protocol has to be adopted by any class that handles EMV authorization responses.

5.3.2 Method Documentation

5.3.2.1 - (void) paymentInfo: (WPPaymentInfo ∗) *paymentInfo* didAuthorize:(WPAuthorizationInfo ∗) *authorizationInfo* `[required]`

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.

5.3.2.2 - (void) paymentInfo: (WPPaymentInfo ∗) *paymentInfo* didFailAuthorization:(NSError ∗) *error* `[required]`

Called when an authorization call fails.

Parameters

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

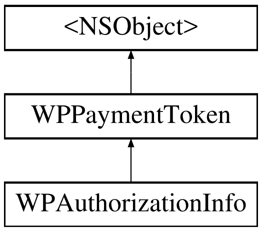
The documentation for this protocol was generated from the following file:

- WePay.h

5.4 WPAuthorizationInfo Class Reference

```
#import <WPAuthorizationInfo.h>
```

Inheritance diagram for WPAuthorizationInfo:



Instance Methods

- (instancetype) - initWithAmount:currencyCode:transactionToken:tokenId:
- (instancetype) - initWithId:

Properties

- NSDecimalNumber ∗ amount
- NSString ∗ currencyCode
- NSString ∗ transactionToken
- NSString ∗ tokenId

5.4.1 Detailed Description

A WPAuthorizationInfo represents authorization information that was obtained from the user's EMV card and is stored on WePay's servers. This information can be used to complete the payment transaction via WePay's web APIs.

5.4.2 Method Documentation

5.4.2.1 - (instancetype) initWithAmount: (NSDecimalNumber ∗) *amount* currencyCode:(NSString ∗) *currencyCode* transactionToken:(NSString ∗) *transactionToken* tokenId:(NSString ∗) *tokenId*

Initializes a WPAuthorizationInfo with the info provided.

Parameters

amount	The amount that was authorized.
currencyCode	The currency code that the amount is specified in.
transactionToken	The transaction token that certifies the transaction
tokenId	The ID of the payment token.

Returns

A WPAuthorizationInfo object initialized with the info provided.

5.4.2.2 - (instancetype) initWithId: (NSString ∗) *tokenId*

Initialzes a WPPaymentToken with the Id provided.

Parameters

token←↩ Id	The Id of the token.

Returns

A WPPaymentToken object initialized with the Id provided.

5.4.3 Property Documentation

5.4.3.1 - (NSDecimalNumber∗) amount `[read], [nonatomic], [strong]`

The amount that was authorized.

5.4.3.2 - (NSString∗) currencyCode `[read], [nonatomic], [strong]`

The currency code that the amount is specified in.

5.4.3.3 - (NSString∗) tokenId `[read], [nonatomic], [strong], [inherited]`

The token's id.

5.4.3.4 - (NSString∗) transactionToken `[read], [nonatomic], [strong]`

The transaction token that certifies the transaction.

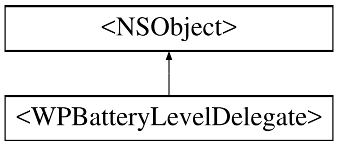
The documentation for this class was generated from the following file:

- WPAuthorizationInfo.h

5.5 <WPBatteryLevelDelegate> Protocol Reference

```
#import <WePay.h>
```

Inheritance diagram for <WPBatteryLevelDelegate>:



Instance Methods

- (void) - didGetBatteryLevel:
- (void) - didFailToGetBatteryLevelwithError:

5.5.1 Detailed Description

This delegate protocol has to be adopted by any class that handles Battery Level responses.

5.5.2 Method Documentation

5.5.2.1 - (void) didFailToGetBatteryLevelwithError: (NSError $*$) *error*

Called when we fail to determine the card reader's battery level.

Parameters

error	The error which caused the failure.

5.5.2.2 - (void) didGetBatteryLevel: (int) *batteryLevel*

Called when the card reader's battery level is determined.

Parameters

batteryLevel	The card reader's battery charge level (0-100%).

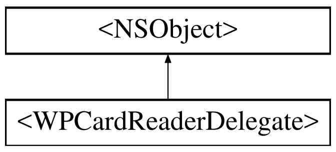
The documentation for this protocol was generated from the following file:

- WePay.h

5.6 <WPCardReaderDelegate> Protocol Reference

```
#import <WePay.h>
```

Inheritance diagram for <WPCardReaderDelegate>:



Instance Methods

- (void) - selectEMVApplication:completion:
- (void) - didReadPaymentInfo:
- (void) - didFailToReadPaymentInfoWithError:
- (void) - selectCardReader:completion:
- (void) - cardReaderDidChangeStatus:
- (void) - shouldResetCardReaderWithCompletion:
- (void) - authorizeAmountWithCompletion:

5.6.1 Detailed Description

This delegate protocol has to be adopted by any class that handles Card Reader responses.

5.6.2 Method Documentation

5.6.2.1 - (void) authorizeAmountWithCompletion: (void(^)(NSDecimalNumber *amount, NSString *currencyCode, long accountId)) *completion* `[optional]`

Called when an EMV reader is connected, so that you can provide the amount, currency code and the WePay account Id of the merchant. The transaction cannot proceed until the completion block is executed. 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, 122.61, 24.61, 124.61, 25.61, and 125.61 will simulate successful auth. Example: completion([NS↩ DecimalNumber decimalNumberWithString:"21.61"], kWPCurrencyCodeUSD, 1234567);

Parameters

completion	The block to be executed with the amount, currency code and merchant account Id for the transaction.
amount	The amount for the transaction. For USD amounts, there can be a maximum of two places after the decimal point. (amount.decimalValue._exponent must be >= -2)
currencyCode	The 3-character ISO 4217 currency code. The only supported currency code is kWPCurrencyCodeUSD.
accountId	The WePay account id of the merchant.

5.6.2.2 - (void) cardReaderDidChangeStatus: (id) *status* `[optional]`

Called when the card reader changes status.

Parameters

status	Current status of the card reader, one of: kWPCardReaderStatusSearching; kWPCardReaderStatusNotConnected; kWPCardReaderStatusConnected; kWPCardReaderStatusCheckingReader; kWPCardReaderStatusConfiguringReader; kWPCardReaderStatusWaitingForCard; kWPCardReaderStatusShouldNotSwipeEMVCard; kWPCardReaderStatusChipErrorSwipeCard; kWPCardReaderStatusSwipeDetected; kWPCardReaderStatusCardDipped; kWPCardReaderStatusTokenizing; kWPCardReaderStatusAuthorizing; kWPCardReaderStatusStopped;

5.6.2.3 - (void) didFailToReadPaymentInfoWithError: (NSError ∗) *error* `[required]`

Called when an error occurs while reading a card.

Parameters

error	The error which caused the failure.

5.6.2.4 - (void) didReadPaymentInfo: (WPPaymentInfo ∗) *paymentInfo* `[required]`

Called when payment info is successfully obtained from a card.

Parameters

paymentInfo	The payment info.

5.6.2.5 - (void) selectCardReader: (NSArray ∗) *cardReaderNames* completion:(void(^)(NSInteger selectedIndex)) *completion* `[required]`

Called after detecting eligible card readers. Either present this card reader list to the merchant, or make some internal default choice. The transaction cannot proceed until the completion block is executed. Example: completion(0);

Parameters

cardreaderNames	The list of detected card readers. Possible entries include "AUDIOJACK" or "MOB30∗", where '∗' indicates the last part of the card reader's serial number found on the back of the device.
completion	The block to be executed with the index of the selected card reader.
selectedIndex	The index of the selected card reader in the array of cardReaderNames.

5.6.2.6 - (void) selectEMVApplication: (NSArray ∗) *applications* completion:(void(^)(NSInteger selectedIndex)) *completion* `[required]`

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, you need to execute the completion block with the index of the selected application. The transaction cannot proceed until the completion block is executed. Example: completion(0);

Parameters

applications	The array of NSStrings containing application names from the card.
completion	The block to be executed with the index of the selected application.
selectedIndex	The index of the selected application in the array of applications from the card.

5.6.2.7 - (void) shouldResetCardReaderWithCompletion: (void(^)(BOOL shouldReset)) *completion* `[optional]`

Optionally called when the connected card reader is already configured, to give the app an opportunity to reset the device. If this method is implemented, the transaction cannot proceed until the completion block is executed. The card reader must be reset here if the merchant manually resets the reader via the hardware reset button on the reader. Examples: completion(YES); completion(NO);

Parameters

completion	The block to be executed with the answer to the question: "Should the card reader be reset?".
shouldReset	The answer to the question: "Should the card reader be reset?".

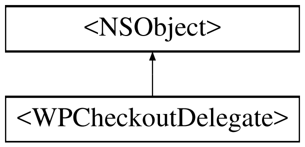
The documentation for this protocol was generated from the following file:

- WePay.h

5.7 <WPCheckoutDelegate> Protocol Reference

```
#import <WePay.h>
```

Inheritance diagram for <WPCheckoutDelegate>:



Instance Methods

- (void) - didStoreSignature:forCheckoutId:
- (void) - didFailToStoreSignatureImage:forCheckoutId:withError:

5.7.1 Detailed Description

This delegate protocol has to be adopted by any class that handles Checkout responses.

5.7.2 Method Documentation

5.7.2.1 - (void) didFailToStoreSignatureImage: (UIImage ∗) *image* forCheckoutId:(NSString ∗) *checkoutId* withError:(NSError ∗) *error*

Called when an error occurs while storing a signature.

Parameters

image	The signature image to be stored.
checkout↩ Id	The checkout id associated with the signature.
error	The error which caused the failure.

5.7.2.2 - (void) didStoreSignature: (NSString ∗) *signatureUrl* forCheckoutId:(NSString ∗) *checkoutId*

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.

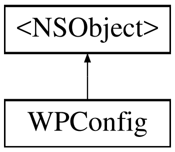
The documentation for this protocol was generated from the following file:

- WePay.h

5.8 WPConfig Class Reference

```
#import <WPConfig.h>
```

Inheritance diagram for WPConfig:



Instance Methods

- (instancetype) - initWithClientId:environment:
- (instancetype) - initWithClientId:environment:useLocation:useTestEMVCards:callDelegateMethodsOnMain↩ Thread:restartTransactionAfterSuccess:restartTransactionAfterGeneralError:restartTransactionAfterOther↩ Errors:stopCardReaderAfterOperation:logLevel:

Properties

- NSString ∗ clientId
- NSString ∗ environment
- BOOL useLocation
- BOOL useTestEMVCards
- BOOL callDelegateMethodsOnMainThread
- BOOL restartTransactionAfterSuccess
- BOOL restartTransactionAfterGeneralError
- BOOL restartTransactionAfterOtherErrors
- BOOL stopCardReaderAfterOperation
- NSString ∗ logLevel
- WPMockConfig ∗ mockConfig

5.8.1 Detailed Description

The configuration object used for initializing a WePay instance.

5.8.2 Method Documentation

5.8.2.1 - (instancetype) initWithClientId: (NSString ∗) *clientId* environment:(NSString ∗) *environment*

A convenience initializer

Parameters

clientId	Your WePay clientId.
environment	The environment to be used, one of (kWPEnvironmentStage, kWPEnvironmentProduction).

Returns

A WPConfig instance which can be used to initialize a WePay instance.

5.8.2.2 - (instancetype) initWithClientId: (NSString ∗) *clientId* environment:(NSString ∗) *environment* useLocation:(BOOL) *useLocation* useTestEMVCards:(BOOL) *useTestEMVCards* callDelegateMethodsOnMainThread:(BOOL) *callDelegateMethodsOnMainThread* restartTransactionAfterSuccess:(BOOL) *restartTransactionAfterSuccess* restartTransactionAfterGeneralError:(BOOL) *restartTransactionAfterGeneralError* restartTransactionAfterOtherErrors:(BOOL) *restartTransactionAfterOtherErrors* stopCardReaderAfterOperation:(BOOL) *stopCardReaderAfterOperation* logLevel:(NSString ∗) *logLevel*

The designated initializer

Parameters

clientId	Your WePay clientId.
environment	The environment to be used, one of (kWPEnvironmentStage, kWPEnvironmentProduction).

Parameters

useLocation	Flag to determine if we should use location services.
useTestEMVCards	Flag to determine if we should use test EMV cards.
callDelegateMethodsOnMainThread	Flag to determine if delegate methods should be called on the main(UI) thread.
restartTransactionAfterSuccess	Flag to determine if the transaction should automatically restart after a successful read.
restartTransactionAfterGeneralError	Flag to determine if the transaction should automatically restart after a general error (domain:kWPErrorCategoryCardReader, errorCode:WPErrorCardReaderGeneralError).
restartTransactionAfterOtherErrors	Flag to determine if the transaction should automatically restart after an error other than general error.
stopCardReaderAfterOperation	Flag to determine if the card reader should automatically stop after an operation is completed.

Returns

A WPConfig instance which can be used to initialize a WePay instance.

5.8.3 Property Documentation

5.8.3.1 - (BOOL) callDelegateMethodsOnMainThread `[read], [write], [nonatomic], [assign]`

Determines if delegate methods should be called on the main(UI) thread. If set to NO, delegate methods will be called on a new background thread. Defaults to YES.

5.8.3.2 - (NSString∗) clientId `[read], [nonatomic], [strong]`

Your WePay clientId for the specified environment

5.8.3.3 - (NSString∗) environment `[read], [nonatomic], [strong]`

The environment to be used, one of (staging, production)

5.8.3.4 - (NSString∗) logLevel `[read], [write], [nonatomic], [strong]`

The log level to be used, one of (all, none). Defaults to kWPLogLevelAll.

5.8.3.5 - (WPMockConfig∗) mockConfig `[read], [write], [nonatomic], [strong]`

The configuration for using mock card reader and/or mock WepayClient implementation

5.8.3.6 - (BOOL) restartTransactionAfterGeneralError `[read], [write], [nonatomic], [assign]`

Determines if the transaction should automatically restart after a swipe/dip general error (domain:kWPErrorCategory↩CardReader, errorCode:WPErrorCardReaderGeneralError). Defaults to YES.

5.8.3.7 - (BOOL) restartTransactionAfterOtherErrors `[read], [write], [nonatomic], [assign]`

Determines if the transaction should automatically restart after a swipe/dip error other than general error. Defaults to NO.

5.8.3.8 - (BOOL) restartTransactionAfterSuccess `[read], [write], [nonatomic], [assign]`

Determines if the transaction should automatically restart after a successful swipe. The transaction is not restarted after a successful dip. Defaults to NO.

5.8.3.9 - (BOOL) stopCardReaderAfterOperation `[read], [write], [nonatomic], [assign]`

Determines if the card reader should automatically stop after an operation is completed. Defaults to YES.

5.8.3.10 - (BOOL) useLocation `[read], [write], [nonatomic], [assign]`

Determines if we should use location services. Defaults to NO.

5.8.3.11 - (BOOL) useTestEMVCards `[read], [write], [nonatomic], [assign]`

Determines if the card reader should accept test EMV cards. Defaults to NO. This should never be turned on in production.

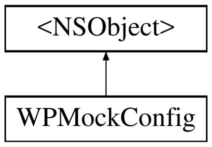
The documentation for this class was generated from the following file:

- WPConfig.h

5.9 WPMockConfig Class Reference

```
#import <WPMockConfig.h>
```

Inheritance diagram for WPMockConfig:



Properties

- BOOL useMockCardReader
- BOOL useMockWepayClient
- BOOL cardReadTimeOut
- BOOL cardReadFailure
- BOOL cardTokenizationFailure
- BOOL EMVAuthFailure
- BOOL multipleEMVApplication
- BOOL batteryLevelError
- BOOL mockCardReaderIsDetected
- NSString ∗ mockPaymentMethod

5.9.1 Detailed Description

The Class MockConfig contains the configuration required when using mock card reader and/or WPClient implementation.

5.9.2 Property Documentation

5.9.2.1 - (BOOL) batteryLevelError `[read], [write], [atomic]`

Determines if a battery info failure should be mocked. Defaults to NO.

5.9.2.2 - (BOOL) cardReadFailure `[read], [write], [atomic]`

Determines if a card reading failure should be mocked. Defaults to NO.

5.9.2.3 - (BOOL) cardReadTimeOut `[read], [write], [atomic]`

Determines if a card reader timeout should be mocked. Defaults to NO.

5.9.2.4 - (BOOL) cardTokenizationFailure `[read], [write], [atomic]`

Determines if a card tokenization failure should be mocked. Defaults to NO.

5.9.2.5 - (BOOL) EMVAuthFailure `[read], [write], [atomic]`

Determines if an EMV authorization failure should be mocked. Defaults to NO.

5.9.2.6 - (BOOL) mockCardReaderIsDetected `[read], [write], [atomic]`

Determines if the mock card reader is available for the purpose of establishing a connection. Defaults to YES.

5.9.2.7 - (NSString$*$) mockPaymentMethod `[read], [write], [atomic]`

The payment method to mock. Defaults to kWPPaymentMethodSwipe.

5.9.2.8 - (BOOL) multipleEMVApplication `[read], [write], [atomic]`

Determines if multiple EMV application should be mocked. Dafaults to NO.

5.9.2.9 - (BOOL) useMockCardReader `[read], [write], [atomic]`

Determines if mock card reader implementation is used. Defaults to YES.

5.9.2.10 - (BOOL) useMockWepayClient `[read], [write], [atomic]`

e Determines if mock WepayClient implementation is used. Defaults to YES.

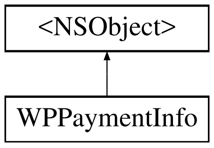
The documentation for this class was generated from the following file:

- WPMockConfig.h

5.10 WPPaymentInfo Class Reference

`#import <WPPaymentInfo.h>`

Inheritance diagram for WPPaymentInfo:



Instance Methods

- (instancetype) - **initWithSwipedInfo:**
- (instancetype) - **initWithEMVInfo:**
- (instancetype) - initWithFirstName:lastName:email:billingAddress:shippingAddress:cardNumber:cvv:expMonth↩ :expYear:virtualTerminal:
- (void) - addEmail:

Properties

- NSString ∗ firstName
- NSString ∗ lastName
- NSString ∗ email
- NSString ∗ paymentDescription
- BOOL isVirtualTerminal
- WPAddress ∗ billingAddress
- WPAddress ∗ shippingAddress
- id paymentMethod
- id swiperInfo
- id manualInfo
- id emvInfo

5.10.1 Detailed Description

An instance of this class represents the payment information obtained from the user via any of the supported payment methods. It is used as input for tokenization operations.

5.10.2 Method Documentation

5.10.2.1 - (void) addEmail: (NSString ∗) *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 address to be added

5.10.2.2 - (instancetype) initWithFirstName: (NSString ∗) *firstName* lastName:(NSString ∗) *lastName* email:(NSString ∗) *email* billingAddress:(WPAddress ∗) *billingAddress* shippingAddress:(WPAddress ∗) *shippingAddress* cardNumber:(NSString ∗) *cardNumber* cvv:(NSString ∗) *cvv* expMonth:(NSString ∗) *expMonth* expYear:(NSString ∗) *expYear* virtualTerminal:(BOOL) *virtualTerminal*

Initializes a WPPaymentInfo instance of type kWPPaymentMethodManual.

Parameters

firstName	First name of the payer.
lastName	Last name of the payer.
email	Email address of the payer.
billingAddress	Billing address.
shippingAddress	Shipping address.
cardNumber	The card number.
cvv	The cvv code.
expMonth	The 2-digit expiration month on the credit card.
expYear	The 4-digit expiration year on the credit card.
virtualTerminal	The virtual terminal flag - should be false if payment info was collected on the payer's device.

Returns

A WPPaymentInfo object initialized with manually obtained card info.

5.10.3 Property Documentation

5.10.3.1 - (WPAddress∗) billingAddress `[read], [nonatomic], [strong]`

Billing address.

5.10.3.2 - (NSString∗) email `[read], [nonatomic], [strong]`

Email address of the payer.

5.10.3.3 - (id) emvInfo `[read], [nonatomic], [strong]`

Additional info obtained by using the EMV payment method.

5.10.3.4 - (NSString∗) firstName `[read], [nonatomic], [strong]`

First name of the payer.

5.10.3.5 - (BOOL) isVirtualTerminal `[read], [nonatomic], [assign]`

Determines if the card was obtained in virtual terminal mode.

5.10.3.6 - (NSString∗) lastName `[read], [nonatomic], [strong]`

Last name of the payer.

5.10.3.7 - (id) manualInfo `[read], [nonatomic], [strong]`

Additional info obtained by using the Manual payment method.

5.10.3.8 - (NSString∗) paymentDescription `[read], [nonatomic], [strong]`

Masked representation of the payment instrument. e.g. XXXXXXXXXXXX1234 Note: the display format may change depending on the payment instrument and the payment method, so this field should not be parsed. It is meant for display to the end user as-is.

5.10.3.9 - (id) paymentMethod `[read], [nonatomic], [strong]`

The payment method used, one of (kWPPaymentMethodManual, kWPPaymentMethodSwipe, kWPPaymentMethod↩ Dip).

5.10.3.10 - (WPAddress∗) shippingAddress `[read], [nonatomic], [strong]`

Shipping address.

5.10.3.11 - (id) swiperInfo `[read], [nonatomic], [strong]`

Additional info obtained by using the Swipe payment method.

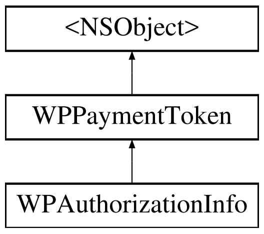
The documentation for this class was generated from the following file:

- WPPaymentInfo.h

5.11 WPPaymentToken Class Reference

```
#import <WPPaymentToken.h>
```

Inheritance diagram for WPPaymentToken:



Instance Methods

- (instancetype) - initWithId:

Properties

- NSString ∗ tokenId

5.11.1 Detailed Description

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

5.11.2 Method Documentation

5.11.2.1 - (instancetype) initWithId: (NSString ∗) *tokenId*

Initialzes a WPPaymentToken with the Id provided.

Parameters

token←↩ Id	The Id of the token.

Returns

A WPPaymentToken object initialized with the Id provided.

5.11.3 Property Documentation

5.11.3.1 - (NSString∗) tokenId `[read], [nonatomic], [strong]`

The token's id.

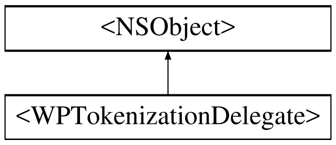
The documentation for this class was generated from the following file:

- WPPaymentToken.h

5.12 <WPTokenizationDelegate> Protocol Reference

```
#import <WePay.h>
```

Inheritance diagram for <WPTokenizationDelegate>:



Instance Methods

- (void) - paymentInfo:didTokenize:
- (void) - paymentInfo:didFailTokenization:
- (void) - insertPayerEmailWithCompletion:

5.12.1 Detailed Description

This delegate protocol has to be adopted by any class that handles tokenization responses.

5.12.2 Method Documentation

5.12.2.1 - (void) insertPayerEmailWithCompletion: (void(^)(NSString *email)) *completion* `[optional]`

Optionally called so that an email address can be provided before a transaction is authorized. If this method is implemented, the transaction cannot proceed until the completion block is executed. Examples: completion("`api@wepay.com`"); completion(nil);

Parameters

completion	The block to be executed with the payer's email address.
email	The payer's email address.

5.12.2.2 - (void) paymentInfo: (WPPaymentInfo *) *paymentInfo* didFailTokenization:(NSError *) *error*

Called when a tokenization call fails.

Parameters

paymentInfo	The payment info that failed tokenization.
error	The error which caused the failure.

5.12.2.3 - (void) paymentInfo: (WPPaymentInfo ∗) *paymentInfo* didTokenize:(WPPaymentToken ∗) *paymentToken*

Called when a tokenization call succeeds.

Parameters

paymentInfo	The payment info that was tokenized.
paymentToken	The payment token representing the payment info.

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

- WePay.h

6 File Documentation

6.1 WPError.h File Reference

WPError.h serves as documentation for all errors surfaced by the WePay iOS SDK.

```
#import <Foundation/Foundation.h>
```

Macros

- #define WPUnexpectedErrorMessage NSLocalizedStringFromTable(@"There was an unexpected error.", @"WePay", @"There was an unexpected error.");
- #define WPNoDataReturnedErrorMessage NSLocalizedStringFromTable(@"There was no data returned.", @"WePay", @"There was no data returned.");
- #define WPCardReaderGeneralErrorMessage NSLocalizedStringFromTable(@"Swipe failed due to: (a) uneven swipe speed, (b) fast swipe, (c) slow swipe, or (d) damaged card.", @"WePay", @"Swipe failed due to: (a) uneven swipe speed, (b) fast swipe, (c) slow swipe, or (d) damaged card.");
- #define WPCardReaderInitializationErrorMessage NSLocalizedStringFromTable(@"Failed to initialize card reader.", @"WePay", @"Failed to initialize card reader.");
- #define WPCardReaderTimeoutErrorMessage NSLocalizedStringFromTable(@"Card reader timed out.", @"WePay", @"Card reader timed out.");
- #define WPSignatureInvalidImageErrorMessage NSLocalizedStringFromTable(@"Inavlid signature image provided.", @"WePay", @"Inavlid signature image provided.");
- #define WPNameNotFoundErrorMessage NSLocalizedStringFromTable(@"Name not found.", @"WePay", @"Name not found.");
- #define WPInvalidCardDataErrorMessage NSLocalizedStringFromTable(@"Invalid card data.", @"WePay", @"Invalid card data.");
- #define WPCardNotSupportedErrorMessage NSLocalizedStringFromTable(@"This card is not supported.", @"WePay", @"This card is not supported.");
- #define WPInvalidApplicationIdErrorMessage NSLocalizedStringFromTable(@"Invalid application ID selected.", @"WePay", @"Invalid application ID selected.");

- #define WPDeclinedByCardErrorMessage NSLocalizedStringFromTable(@"The transaction was declined by the card.", @"WePay", @"The transaction was declined by the card.");
- #define WPCardBlockedErrorMessage NSLocalizedStringFromTable(@"This card has been blocked.", @"We↩ Pay", @"This card has been blocked.");
- #define WPDeclinedByIssuerErrorMessage NSLocalizedStringFromTable(@"The transaction was declined by the issuer bank.", @"WePay", @"The transaction was declined by the issuer bank.");
- #define WPIssuerUnreachableErrorMessage NSLocalizedStringFromTable(@"The issuing bank could not be reached.", @"WePay", @"The issuing bank could not be reached.");
- #define WPInvalidAuthInfoErrorMessage NSLocalizedStringFromTable(@"The provided auth info is invalid.", @"WePay", @"The provided auth info is invalid.");
- #define WPAuthInfoNotProvidedErrorMessage NSLocalizedStringFromTable(@"Auth info was not provided.", @"WePay", @"Auth info was not provided.");
- #define WPPaymentMethodCannotBeTokenizedErrorMessage NSLocalizedStringFromTable(@"This payment method cannot be tokenized.", @"WePay", @"This payment method cannot be tokenized.");
- #define WPFailedToGetBatteryLevelErrorMessage NSLocalizedStringFromTable(@"Battery level could not be determined.", @"WePay", @"Battery level could not be determined.");
- #define WPCardReaderNotConnectedErrorMessage NSLocalizedStringFromTable(@"Card reader is not connected.", @"WePay", @"Card reader is not connected.");
- #define WPCardReaderModelNotSupportedErrorMessage NSLocalizedStringFromTable(@"This card reader model is not supported.", @"WePay", @"This card reader model is not supported.");
- #define WPErrorInvalidTransactionAmountErrorMessage NSLocalizedStringFromTable(@"The provided transaction amount is invalid.", @"WePay", @"The provided transaction amount is invalid.");
- #define WPErrorInvalidTransactionCurrencyCodeErrorMessage NSLocalizedStringFromTable(@"The provided currency code is invalid.", @"WePay", @"The provided currency code is invalid.");
- #define WPErrorInvalidTransactionAccountIDErrorMessage NSLocalizedStringFromTable(@"The provided account ID is invalid.", @"WePay", @"The provided account ID is invalid.");
- #define WPErrorInvalidCardReaderSelectionErrorMessage NSLocalizedStringFromTable(@"Card reader selection is invalid.", @"WePay", @"Card reader selection is invalid.");
- #define WPErrorCardReaderBatteryTooLowErrorMessage NSLocalizedStringFromTable(@"The card reader battery does not have enough charge. Please charge before using.", @"WePay", @"The card reader battery does not have enough charge. Please charge before using.");
- #define WPErrorCardReaderUnableToConnectErrorMessage NSLocalizedStringFromTable(@"Please make sure you're using a supported card reader and that it is fully charged.", @"WePay", @"Please make sure you're using a supported card reader and that it is fully charged.");

Enumerations

- enum WPErrorCode {
  WPErrorUnknown = -10000, WPErrorNoDataReturned = -10015, WPErrorCardReaderGeneralError = -10016, WPErrorCardReaderInitialization = -10017,
  WPErrorCardReaderTimeout = -10018, WPErrorCardReaderStatusError = -10019, WPErrorInvalidSignature↩ Image = -10020, WPErrorNameNotFound = -10021,
  WPErrorInvalidCardData = -10022, WPErrorCardNotSupported = -10023, WPErrorEMVTransactionError = -10024, WPErrorInvalidApplicationId = -10025,
  WPErrorDeclinedByCard = -10026, WPErrorCardBlocked = -10027, WPErrorDeclinedByIssuer = -10028, WP↩ ErrorIssuerUnreachable = -10029,
  WPErrorInvalidAuthInfo = -10030, WPErrorAuthInfoNotProvided = -10031, WPErrorPaymentMethodCannotBe↩ Tokenized = -10032, WPErrorFailedToGetBatteryLevel = -10033,
  WPErrorCardReaderNotConnected = -10034, WPErrorCardReaderModelNotSupported = -10035, WPError↩ InvalidTransactionAmount = -10036, WPErrorInvalidTransactionCurrencyCode = -10037,
  WPErrorInvalidTransactionAccountID = -10038, WPErrorInvalidCardReaderSelection = -10039, WPErrorCard↩ ReaderBatteryTooLow = -10040, WPErrorCardReaderUnableToConnect = -10041 }

Variables

- FOUNDATION_EXPORT NSString ∗const kWPErrorAPIDomain
- FOUNDATION_EXPORT NSString ∗const kWPErrorSDKDomain
- FOUNDATION_EXPORT NSString ∗const kWPErrorCategoryKey
- FOUNDATION_EXPORT NSString ∗const kWPErrorCategoryNone
- FOUNDATION_EXPORT NSString ∗const kWPErrorCategoryCardReader
- FOUNDATION_EXPORT NSString ∗const kWPErrorCategoryCardSDK
- enum WPErrorCode **WPErrorCode**

6.1.1 Detailed Description

WPError.h serves as documentation for all errors surfaced by the WePay iOS SDK.

When errors occur, the WePay iOS SDK returns NSError instances to delegate methods. Each error instance has the following components:

- [error code] gives the integer code corresponding with the error
- [error domain] gives the domain that the error belongs to
- [error userInfo] gives a dictionary with some more useful info, which can be accessed with the keys kWPError←CategoryKey and NSLocalizedDescriptionKey

The WePay iOS SDK can return errors in various error domains:

- WePay server API errors are in the kWPErrorAPIDomain
- Errors generated by the SDK itself are in the kWPErrorSDKDomain
- System errors generated by iOS are passed through as-is, for example in the NSURLErrorDomain

See the WPErrorCode section for more details about error codes.

6.1.2 Macro Definition Documentation

6.1.2.1 #define WPAuthInfoNotProvidedErrorMessage NSLocalizedStringFromTable(@"Auth info was not provided.", @"WePay", @"Auth info was not provided.");

The localizable user facing message for WPErrorAuthInfoNotProvided, that can be retrieved by calling [error localized←Description].

6.1.2.2 #define WPCardBlockedErrorMessage NSLocalizedStringFromTable(@"This card has been blocked.", @"WePay", @"This card has been blocked.");

The localizable user facing message for WPErrorCardBlocked, that can be retrieved by calling [error localized←Description].

6.1.2.3 #define WPCardNotSupportedErrorMessage NSLocalizedStringFromTable(@"This card is not supported.", @"WePay", @"This card is not supported.");

The localizable user facing message for WPErrorCardNotSupported, that can be retrieved by calling [error localized←Description].

6.1.2.4 #define WPCardReaderGeneralErrorMessage NSLocalizedStringFromTable(@"Swipe failed due to: (a) uneven swipe speed, (b) fast swipe, (c) slow swipe, or (d) damaged card.", @"WePay", @"Swipe failed due to: (a) uneven swipe speed, (b) fast swipe, (c) slow swipe, or (d) damaged card.");

The localizable user facing message for WPErrorCardReaderGeneralError, that can be retrieved by calling [error localizedDescription].

6.1.2.5 #define WPCardReaderInitializationErrorMessage NSLocalizedStringFromTable(@"Failed to initialize card reader.", @"WePay", @"Failed to initialize card reader.");

The localizable user facing message for WPErrorCardReaderInitialization, that can be retrieved by calling [error localizedDescription].

6.1.2.6 #define WPCardReaderModelNotSupportedErrorMessage NSLocalizedStringFromTable(@"This card reader model is not supported.", @"WePay", @"This card reader model is not supported.");

The localizable user facing message for WPErrorCardReaderModelNotSupported, that can be retrieved by calling [error localizedDescription].

6.1.2.7 #define WPCardReaderNotConnectedErrorMessage NSLocalizedStringFromTable(@"Card reader is not connected.", @"WePay", @"Card reader is not connected.");

The localizable user facing message for WPErrorCardReaderNotConnected, that can be retrieved by calling [error localizedDescription].

6.1.2.8 #define WPCardReaderTimeoutErrorMessage NSLocalizedStringFromTable(@"Card reader timed out.", @"WePay", @"Card reader timed out.");

The localizable user facing message for WPErrorCardReaderTimeout, that can be retrieved by calling [error localized←Description].

6.1.2.9 #define WPDeclinedByCardErrorMessage NSLocalizedStringFromTable(@"The transaction was declined by the card.", @"WePay", @"The transaction was declined by the card.");

The localizable user facing message for WPErrorDeclinedByCard, that can be retrieved by calling [error localized←Description].

6.1.2.10 #define WPDeclinedByIssuerErrorMessage NSLocalizedStringFromTable(@"The transaction was declined by the issuer bank.", @"WePay", @"The transaction was declined by the issuer bank.");

The localizable user facing message for WPErrorDeclinedByIssuer, that can be retrieved by calling [error localized←Description].

6.1.2.11 #define WPErrorCardReaderBatteryTooLowErrorMessage NSLocalizedStringFromTable(@"The card reader battery does not have enough charge. Please charge before using.", @"WePay", @"The card reader battery does not have enough charge. Please charge before using.");

The localizable user facing message for WPErrorCardReaderBatteryTooLow, that can be retrieved by calling [error localizedDescription].

6.1.2.12 #define WPErrorCardReaderUnableToConnectErrorMessage NSLocalizedStringFromTable(@"Please make sure you're using a supported card reader and that it is fully charged.", @"WePay", @"Please make sure you're using a supported card reader and that it is fully charged.");

The localizable user facing message for WPErrorCardReaderUnsupportedOrBatteryTooLow, that can be retrieved by calling [error localizedDescription].

6.1.2.13 #define WPErrorInvalidCardReaderSelectionErrorMessage NSLocalizedStringFromTable(@"Card reader selection is invalid.", @"WePay", @"Card reader selection is invalid.");

The localizable user facing message for WPErrorInvalidCardReaderSelection, that can be retrieved by calling [error localizedDescription].

6.1.2.14 #define WPErrorInvalidTransactionAccountIDErrorMessage NSLocalizedStringFromTable(@"The provided account ID is invalid.", @"WePay", @"The provided account ID is invalid.");

The localizable user facing message for WPErrorInvalidTransactionAccountID, that can be retrieved by calling [error localizedDescription].

6.1.2.15 #define WPErrorInvalidTransactionAmountErrorMessage NSLocalizedStringFromTable(@"The provided transaction amount is invalid.", @"WePay", @"The provided transaction amount is invalid.");

The localizable user facing message for WPErrorInvalidTransactionAmount, that can be retrieved by calling [error localizedDescription].

6.1.2.16 #define WPErrorInvalidTransactionCurrencyCodeErrorMessage NSLocalizedStringFromTable(@"The provided currency code is invalid.", @"WePay", @"The provided currency code is invalid.");

The localizable user facing message for WPErrorInvalidTransactionCurrencyCode, that can be retrieved by calling [error localizedDescription].

6.1.2.17 #define WPFailedToGetBatteryLevelErrorMessage NSLocalizedStringFromTable(@"Battery level could not be determined.", @"WePay", @"Battery level could not be determined.");

The localizable user facing message for WPErrorFailedToGetBatteryLevel, that can be retrieved by calling [error localizedDescription].

6.1.2.18 #define WPInvalidApplicationIdErrorMessage NSLocalizedStringFromTable(@"Invalid application ID selected.", @"WePay", @"Invalid application ID selected.");

The localizable user facing message for WPErrorInvalidApplicationId, that can be retrieved by calling [error localized↩ Description].

6.1.2.19 #define WPInvalidAuthInfoErrorMessage NSLocalizedStringFromTable(@"The provided auth info is invalid.", @"WePay", @"The provided auth info is invalid.");

The localizable user facing message for WPErrorInvalidAuthInfo, that can be retrieved by calling [error localized↩ Description].

6.1.2.20 #define WPInvalidCardDataErrorMessage NSLocalizedStringFromTable(@"Invalid card data.", @"WePay", @"Invalid card data.");

The localizable user facing message for WPErrorInvalidCardData, that can be retrieved by calling [error localized↩ Description].

6.1.2.21 #define WPIssuerUnreachableErrorMessage NSLocalizedStringFromTable(@"The issuing bank could not be reached.", @"WePay", @"The issuing bank could not be reached.");

The localizable user facing message for WPErrorIssuerUnreachable, that can be retrieved by calling [error localized↩ Description].

6.1.2.22 #define WPNameNotFoundErrorMessage NSLocalizedStringFromTable(@"Name not found.", @"WePay", @"Name not found.");

The localizable user facing message for WPErrorNameNotFound, that can be retrieved by calling [error localized↩ Description].

6.1.2.23 #define WPNoDataReturnedErrorMessage NSLocalizedStringFromTable(@"There was no data returned.", @"WePay", @"There was no data returned.");

The localizable user facing message for WPErrorNoDataReturned, that can be retrieved by calling [error localized↩ Description].

6.1.2.24 #define WPPaymentMethodCannotBeTokenizedErrorMessage NSLocalizedStringFromTable(@"This payment method cannot be tokenized.", @"WePay", @"This payment method cannot be tokenized.");

The localizable user facing message for WPErrorPaymentMethodCannotBeTokenized, that can be retrieved by calling [error localizedDescription].

6.1.2.25 #define WPSignatureInvalidImageErrorMessage NSLocalizedStringFromTable(@"Inavlid signature image provided.", @"WePay", @"Inavlid signature image provided.");

The localizable user facing message for WPErrorInavlidSignatureImage, that can be retrieved by calling [error localizedDescription].

6.1.2.26 #define WPUnexpectedErrorMessage NSLocalizedStringFromTable(@"There was an unexpected error.", @"WePay", @"There was an unexpected error.");

The localizable user facing message for WPErrorUnknown, that can be retrieved by calling [error localizedDescription].

6.1.3 Enumeration Type Documentation

6.1.3.1 enum WPErrorCode

Error codes for NSErrors surfaced by the WePay iOS SDK in the kWPErrorSDKDomain. For a full list of error codes in the kWPErrorAPIDomain, visit `https://www.wepay.com/developer/reference/errors`

Enumerator

WPErrorUnknown -10000 Unknown error.

WPErrorNoDataReturned -10015 No data returned by the API call.

WPErrorCardReaderGeneralError -10016 General error reported by the card reader - usually due to a bad swipe.

WPErrorCardReaderInitialization -10017 Error while initializing the card reader.

WPErrorCardReaderTimeout -10018 Timeout occurred while waiting for card.

WPErrorCardReaderStatusError -10019 Special error reported by card reader - very rare.

WPErrorInvalidSignatureImage -10020 Invalid signature image.

WPErrorNameNotFound -10021 Name not found.

WPErrorInvalidCardData -10022 Invalid card data.

WPErrorCardNotSupported -10023 Card not supported.

WPErrorEMVTransactionError -10024 EMV transaction error.

WPErrorInvalidApplicationId -10025 Invalid application ID.

WPErrorDeclinedByCard -10026 Declined by card.

WPErrorCardBlocked -10027 Card blocked.

WPErrorDeclinedByIssuer -10028 Declined by issuer.

WPErrorIssuerUnreachable -10029 Issuer unreachable.

WPErrorInvalidAuthInfo -10030 Invalid auth info.

WPErrorAuthInfoNotProvided -10031 Auth info not provided.

WPErrorPaymentMethodCannotBeTokenized -10032 Payment method cannot be tokenized.

WPErrorFailedToGetBatteryLevel -10033 Failed to get battery level.

WPErrorCardReaderNotConnected -10034 Card reader not connected.

WPErrorCardReaderModelNotSupported -10035 Card reader model not supported.

WPErrorInvalidTransactionAmount -10036 Invalid transaction amount.

WPErrorInvalidTransactionCurrencyCode -10037 Invalid transaction currency code.

WPErrorInvalidTransactionAccountID -10038 Invalid transaction account id.

WPErrorInvalidCardReaderSelection -10039 Invalid card reader selection.

WPErrorCardReaderBatteryTooLow -10040 Card reader battery too low.

WPErrorCardReaderUnableToConnect -10041 Unable to connect to card reader.

6.1.4 Variable Documentation

6.1.4.1 FOUNDATION_EXPORT NSString∗ const kWPErrorAPIDomain

The NSError domain of all errors surfaced by the WePay iOS SDK that were returned by the WePay API. For a full list of error codes in the kWPErrorAPIDomain, visit `https://www.wepay.com/developer/reference/errors`

6.1.4.2 FOUNDATION_EXPORT NSString∗ const kWPErrorCategoryCardReader

The value used in the NSError's userInfo dictionary to return the "card reader" error category.

6.1.4.3 FOUNDATION_EXPORT NSString∗ const kWPErrorCategoryCardSDK

The value used in the NSError's userInfo dictionary to return the "sdk" error category.

6.1.4.4 FOUNDATION_EXPORT NSString∗ const kWPErrorCategoryKey

The key used in the NSError's userInfo dictionary to return the error category.

6.1.4.5 FOUNDATION_EXPORT NSString∗ const kWPErrorCategoryNone

The value used in the NSError's userInfo dictionary to return the "none" error category.

6.1.4.6 FOUNDATION_EXPORT NSString∗ const kWPErrorSDKDomain

The NSError domain of all errors returned by the WePay iOS SDK itself. For a full list of error codes in the kWPError↩SDKDomain, look at WPErrorCode.

Index