

WePay iOS SDK
7.0.0-prerelease-1

Generated by Doxygen 1.8.13

Contents

1	Getting Started	2
2	Hierarchical Index	12
2.1	Class Hierarchy	12
3	Class Index	13
3.1	Class List	13
4	File Index	13
4.1	File List	13
5	Class Documentation	14
5.1	WePay Class Reference	14
5.1.1	Detailed Description	15
5.1.2	Method Documentation	15
5.1.3	Property Documentation	18
5.2	WPAAddress Class Reference	19
5.2.1	Detailed Description	19
5.2.2	Method Documentation	20
5.2.3	Property Documentation	21
5.3	<WPAuthorizationDelegate > Protocol Reference	22
5.3.1	Method Documentation	22
5.4	<WPAuthorizationDelegate> Protocol Reference	23
5.4.1	Detailed Description	23
5.5	WPAuthorizationInfo Class Reference	23
5.5.1	Detailed Description	24
5.5.2	Method Documentation	24
5.5.3	Property Documentation	25
5.6	<WPBatteryLevelDelegate > Protocol Reference	26

5.6.1	Method Documentation	26
5.7	<WPBatteryLevelDelegate> Protocol Reference	27
5.7.1	Detailed Description	27
5.8	<WPCardReaderDelegate > Protocol Reference	27
5.8.1	Method Documentation	27
5.9	<WPCardReaderDelegate> Protocol Reference	30
5.9.1	Detailed Description	30
5.10	<WPCheckoutDelegate > Protocol Reference	30
5.10.1	Method Documentation	31
5.11	<WPCheckoutDelegate> Protocol Reference	32
5.11.1	Detailed Description	32
5.12	WPConfig Class Reference	32
5.12.1	Detailed Description	33
5.12.2	Method Documentation	33
5.12.3	Property Documentation	34
5.13	WPMockConfig Class Reference	36
5.13.1	Detailed Description	36
5.13.2	Property Documentation	37
5.14	WPPaymentInfo Class Reference	38
5.14.1	Detailed Description	39
5.14.2	Method Documentation	39
5.14.3	Property Documentation	40
5.15	WPPaymentToken Class Reference	42
5.15.1	Detailed Description	42
5.15.2	Method Documentation	42
5.15.3	Property Documentation	43
5.16	<WPTokenizationDelegate > Protocol Reference	43
5.16.1	Method Documentation	43
5.17	<WPTokenizationDelegate> Protocol Reference	44
5.17.1	Detailed Description	45

6 File Documentation	45
6.1 WPErrors.h File Reference	45
6.1.1 Detailed Description	47
6.1.2 Macro Definition Documentation	47
6.1.3 Enumeration Type Documentation	51
6.1.4 Variable Documentation	52
Index	55

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 processing 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 NSLocationWhenInUseUsageDescription.

```
<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 = kWPCardReader↵StatusSearching`.
- 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 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 `cardReaderDidChange↵Status:` 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 = kWPCardReader↵StatusWaitingForCard`.

6. If the user swipes a card successfully:

- The `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReader↵StatusSwipeDetected`.
- 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↵StatusTokenizing`, 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 = kWPCardReader↵StatusCardDipped`
- 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 `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 = kWPCardReaderStatusAuthorizing`, 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 = kWPCardReaderStatusWaitingForCard` 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 the failure methods will be called with the appropriate error.
 10. When processing stops, the `cardReaderDidChangeStatus:` method will be called with `status = kWPCardReaderStatusStopped`.

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:[
WPAAddress 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 [WPCardReaderDelegate](#) 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:

<NSObject>

WePay	14
WPAddress	19
<WPAuthorizationDelegate >	22
<WPBatteryLevelDelegate >	26
<WPCardReaderDelegate >	27
<WPCheckoutDelegate >	30
WPConfig	32
WPMockConfig	36
WPPaymentInfo	38
WPPaymentToken	42

WPAuthorizationInfo	23
<WPTokenizationDelegate >	43
<WPAuthorizationDelegate>	23
<WPBatteryLevelDelegate>	27
<WPCardReaderDelegate>	30
<WPCheckoutDelegate>	32
<WPTokenizationDelegate>	44

3 Class Index

3.1 Class List

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

WePay	14
WPAAddress	19
<WPAuthorizationDelegate >	22
<WPAuthorizationDelegate>	23
WPAuthorizationInfo	23
<WPBatteryLevelDelegate >	26
<WPBatteryLevelDelegate>	27
<WPCardReaderDelegate >	27
<WPCardReaderDelegate>	30
<WPCheckoutDelegate >	30
<WPCheckoutDelegate>	32
WPConfig	32
WPMockConfig	36
WPPaymentInfo	38
WPPaymentToken	42
<WPTokenizationDelegate >	43
<WPTokenizationDelegate>	44

4 File Index

4.1 File List

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

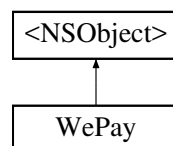
WePay.h	??
WPAAddress.h	??
WPAuthorizationInfo.h	??
WPBatteryHelper.h	??
WPConfig.h	??
WPConstantsExternal.h	??
WPErrors.h	
WPErrors.h serves as documentation for all errors surfaced by the WePay iOS SDK	
	45
WPMockConfig.h	??
WPPaymentInfo.h	??
WPPaymentToken.h	??

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:authorizationDelegate:](#)
- (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

- [WPCConfig](#) * [config](#)

5.1.1 Detailed Description

Main Class containing all public endpoints.

5.1.2 Method Documentation

5.1.2.1 [forgetRememberedCardReader\(\)](#)

```
- (void) forgetRememberedCardReader
```

Clears the name of the most recently used card reader.

5.1.2.2 [getCardReaderBatteryLevelWithCardReaderDelegate:batteryLevelDelegate:\(\)](#)

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

Gets the current battery level of the card reader.

Parameters

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

5.1.2.3 getRememberedCardReader()

```
- (NSString *) getRememberedCardReader
```

Gets the name of the most recently used card reader.

Returns

the name of the card reader.

5.1.2.4 initWithConfig:()

```
- (instancetype) initWithConfig:
    (WPConfig *) config
```

The designated initializer. Use this to initialize the SDK.

Parameters

<i>config</i>	A WPConfig instance.
---------------	--------------------------------------

Returns

A [WePay](#) instance, which can be used to access most of the functionality of this sdk.

5.1.2.5 startTransactionForReadingWithCardReaderDelegate:()

```
- (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 timeout 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:WPErrorCategoryCardReader, 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 kWPCardReaderStatusWaitingForCard, 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

<i>cardReaderDelegate</i>	The delegate class which will receive the response(s) for this call.
---------------------------	--

5.1.2.6 startTransactionForTokenizingWithCardReaderDelegate:tokenizationDelegate:authorizationDelegate:()

```
- (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 timeout 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:WPErrorCategoryCardReader, 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 kWPCardReaderStatusWaitingForCard, 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

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

5.1.2.7 stopCardReader()

```
- (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 storeSignatureImage:forCheckoutId:checkoutDelegate:()

```
- (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

<i>image</i>	The signature image to be stored.
<i>checkoutId</i>	The checkout id associated with this transaction.
<i>checkoutDelegate</i>	The delegate class which will receive the response(s) for this call.

5.1.2.9 tokenizePaymentInfo:tokenizationDelegate:()

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

Creates a payment token from a [WPPaymentInfo](#) object.

Parameters

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

5.1.3 Property Documentation

5.1.3.1 config

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

Your [WePay](#) config

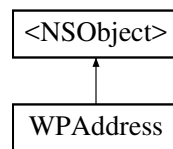
The documentation for this class was generated from the following file:

- [WePay.h](#)

5.2 WPAAddress Class Reference

```
#import <WPAAddress.h>
```

Inheritance diagram for WPAAddress:



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 initWithAddress1:address2:city:region:postcode:country:()

```
- (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

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

Returns

The address.

5.2.2.2 initWithAddress1:address2:city:state:zip:()

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

Initializes a US Address.

Parameters

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

Returns

The address.

5.2.2.3 initWithZip:()

```
- (instancetype) initWithZip:
    (NSString *) zip
```

Initializes a US Address with just a zip.

Parameters

<i>zip</i>	The US zip or zip-plus code.
------------	------------------------------

Returns

The address.

5.2.3 Property Documentation**5.2.3.1 address1**

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

The first line of the street address.

5.2.3.2 address2

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

The second line of the street address.

5.2.3.3 city

- (NSString*) city [read], [nonatomic], [strong]

The city.

5.2.3.4 country

- (NSString*) country [read], [nonatomic], [strong]

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

5.2.3.5 postcode

- (NSString*) postcode [read], [nonatomic], [strong]

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

5.2.3.6 region

- (NSString*) region [read], [nonatomic], [strong]

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

5.2.3.7 state

- (NSString*) state [read], [nonatomic], [strong]

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

5.2.3.8 zip

- (NSString*) zip [read], [nonatomic], [strong]

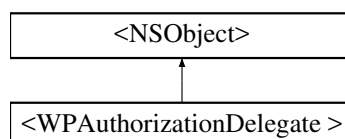
The US zip or zip-plus code. Only for US addresses.

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

- WPAAddress.h

5.3 <WPAuthorizationDelegate > Protocol Reference

Inheritance diagram for <WPAuthorizationDelegate >:



Instance Methods

- (void) - [paymentInfo:didAuthorize:](#)
- (void) - [paymentInfo:didFailAuthorization:](#)

5.3.1 Method Documentation

5.3.1.1 [paymentInfo:didAuthorize:\(\)](#)

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

Called when an authorization call succeeds.

Parameters

<i>paymentInfo</i>	The payment info for the card that was authorized.
<i>authorizationInfo</i>	The authorization info for the transaction that was authorized.

5.3.1.2 [paymentInfo:didFailAuthorization:\(\)](#)

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

Called when an authorization call fails.

Parameters

<i>paymentInfo</i>	The payment info for the card that failed authorization.
<i>error</i>	The error which caused the failure.

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

- [WePay.h](#)

5.4 <WPAuthorizationDelegate> Protocol Reference

```
#include <WePay.h>
```

5.4.1 Detailed Description

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

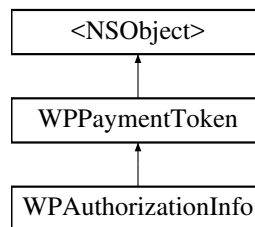
The documentation for this protocol was generated from the following file:

- `WePay.h`

5.5 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](#)

5.5.1 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.

5.5.2 Method Documentation

5.5.2.1 [initWithAmount:currencyCode:transactionToken:tokenId:\(\)](#)

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

Initializes a [WPAuthorizationInfo](#) with the info provided.

Parameters

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

Returns

A [WPAuthorizationInfo](#) object initialized with the info provided.

5.5.2.2 initWithId:()

```
- (instancetype) initWithId:
    (NSString *) tokenId
```

Initializes a [WPPaymentToken](#) with the Id provided.

Parameters

<i>tokenId</i>	The Id of the token.
----------------	----------------------

Returns

A [WPPaymentToken](#) object initialized with the Id provided.

5.5.3 Property Documentation

5.5.3.1 amount

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

The amount that was authorized.

5.5.3.2 currencyCode

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

The currency code that the amount is specified in.

5.5.3.3 tokenId

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

The token's id.

5.5.3.4 transactionToken

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

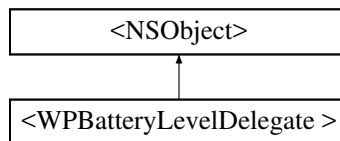
The transaction token that certifies the transaction.

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

- WPAuthorizationInfo.h

5.6 <WPBatteryLevelDelegate > Protocol Reference

Inheritance diagram for <WPBatteryLevelDelegate >:



Instance Methods

- (void) - [didGetBatteryLevel:](#)
- (void) - [didFailToGetBatteryLevelWithError:](#)

5.6.1 Method Documentation

5.6.1.1 didFailToGetBatteryLevelWithError:()

```
- (void WPBatteryLevelDelegate) didFailToGetBatteryLevelWithError:
    (NSError *) error
```

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

Parameters

<i>error</i>	The error which caused the failure.
--------------	-------------------------------------

5.6.1.2 didGetBatteryLevel:()

```
- (void WPBatteryLevelDelegate) didGetBatteryLevel:
    (int) batteryLevel
```

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

Parameters

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

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

- WePay.h

5.7 <WPBatteryLevelDelegate> Protocol Reference

```
#include <WePay.h>
```

5.7.1 Detailed Description

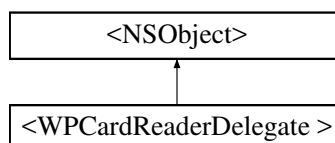
This delegate protocol has to be adopted by any class that handles Battery Level responses.

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

- WePay.h

5.8 <WPCardReaderDelegate > Protocol Reference

Inheritance diagram for <WPCardReaderDelegate >:



Instance Methods

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

5.8.1 Method Documentation

5.8.1.1 [authorizeAmountWithCompletion:\(\)](#)

```
- (void WPCardReaderDelegate) 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([NSNumber numberWithInt:21.61], kWPCurrencyCodeUSD, 1234567);`

Parameters

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

5.8.1.2 [cardReaderDidChangeStatus:\(\)](#)

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

Called when the card reader changes status.

Parameters

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

5.8.1.3 didFailToReadPaymentInfoWithError:()

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

Called when an error occurs while reading a card.

Parameters

<i>error</i>	The error which caused the failure.
--------------	-------------------------------------

5.8.1.4 didReadPaymentInfo:()

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

Called when payment info is successfully obtained from a card.

Parameters

<i>paymentInfo</i>	The payment info.
--------------------	-------------------

5.8.1.5 selectCardReader:completion:()

```
- (void WPCardReaderDelegate) 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

<i>cardreaderNames</i>	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.
<i>completion</i>	The block to be executed with the index of the selected card reader.
<i>selectedIndex</i>	The index of the selected card reader in the array of cardReaderNames.

5.8.1.6 selectEMVApplication:completion:()

```
- (void WPCardReaderDelegate) 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

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

5.8.1.7 shouldResetCardReaderWithCompletion:()

```
- (void WPCardReaderDelegate) 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

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

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

- WePay.h

5.9 <WPCardReaderDelegate> Protocol Reference

```
#include <WePay.h>
```

5.9.1 Detailed Description

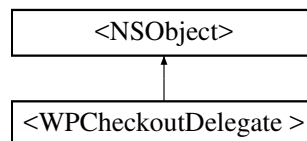
This delegate protocol has to be adopted by any class that handles Card Reader responses.

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

- WePay.h

5.10 <WPCheckoutDelegate> Protocol Reference

Inheritance diagram for <WPCheckoutDelegate>:



Instance Methods

- (void) - [didStoreSignature:forCheckoutId:](#)
- (void) - [didFailToStoreSignatureImage:forCheckoutId:withError:](#)

5.10.1 Method Documentation

5.10.1.1 didFailToStoreSignatureImage:forCheckoutId:withError:()

```
- (void WPCheckoutDelegate) didFailToStoreSignatureImage:
    (UIImage *) image
    forCheckoutId:(NSString *) checkoutId
    withError:(NSError *) error
```

Called when an error occurs while storing a signature.

Parameters

<i>image</i>	The signature image to be stored.
<i>checkoutId</i>	The checkout id associated with the signature.
<i>error</i>	The error which caused the failure.

5.10.1.2 didStoreSignature:forCheckoutId:()

```
- (void WPCheckoutDelegate) didStoreSignature:
    (NSString *) signatureUrl
    forCheckoutId:(NSString *) checkoutId
```

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

Parameters

<i>signatureUrl</i>	The url for the signature image.
<i>checkoutId</i>	The checkout id associated with the signature.

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

- WePay.h

5.11 <WPCheckoutDelegate> Protocol Reference

```
#include <WePay.h>
```

5.11.1 Detailed Description

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

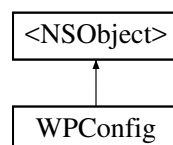
The documentation for this protocol was generated from the following file:

- WePay.h

5.12 WPConfig Class Reference

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

Inheritance diagram for WPConfig:



Instance Methods

- (instancetype) - initWithClientId:environment:
- (instancetype) - initWithClientId:environment:useLocation:useTestEMVCards:callDelegateMethodsOnMainThread:restartTransactionAfterSuccess:restartTransactionAfterGeneralError:restartTransactionAfterOtherErrors: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`

5.12.1 Detailed Description

The configuration object used for initializing a [WePay](#) instance.

5.12.2 Method Documentation

5.12.2.1 initWithClientId:environment:()

```

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

```

A convenience initializer

Parameters

<i>clientId</i>	Your WePay clientId.
<i>environment</i>	The environment to be used, one of (kWPEEnvironmentStage, kWPEEnvironmentProduction).

Returns

A [WPCongig](#) instance which can be used to initialize a [WePay](#) instance.

5.12.2.2 initWithClientId:environment:useLocation:useTestEMVCards:callDelegateMethodsOnMainThread:restartTransactionAfterSuccess:restartTransactionAfterGeneralError:restartTransactionAfterOtherErrors:stopCardReaderAfterOperation:logLevel:()

```
- (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

<i>clientId</i>	Your WePay clientId.
<i>environment</i>	The environment to be used, one of (kWPEnvironmentStage, kWPEnvironmentProduction).
<i>useLocation</i>	Flag to determine if we should use location services.
<i>useTestEMVCards</i>	Flag to determine if we should use test EMV cards.
<i>callDelegateMethodsOnMainThread</i>	Flag to determine if delegate methods should be called on the main(UI) thread.
<i>restartTransactionAfterSuccess</i>	Flag to determine if the transaction should automatically restart after a successful read.
<i>restartTransactionAfterGeneralError</i>	Flag to determine if the transaction should automatically restart after a general error (domain:kWPErrorCategoryCardReader, errorCode:WPErrorCardReaderGeneralError).
<i>restartTransactionAfterOtherErrors</i>	Flag to determine if the transaction should automatically restart after an error other than general error.
<i>stopCardReaderAfterOperation</i>	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.

5.12.3 Property Documentation

5.12.3.1 callDelegateMethodsOnMainThread

```
- (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.

5.12.3.2 clientId

- (NSString*) clientId [read], [nonatomic], [strong]

Your [WePay](#) clientId for the specified environment

5.12.3.3 environment

- (NSString*) environment [read], [nonatomic], [strong]

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

5.12.3.4 logLevel

- (NSString*) logLevel [read], [write], [nonatomic], [strong]

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

5.12.3.5 mockConfig

- (WPMockConfig*) mockConfig [read], [write], [nonatomic], [strong]

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

5.12.3.6 restartTransactionAfterGeneralError

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

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

5.12.3.7 restartTransactionAfterOtherErrors

- (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.

5.12.3.8 restartTransactionAfterSuccess

- (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.

5.12.3.9 stopCardReaderAfterOperation

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

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

5.12.3.10 useLocation

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

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

5.12.3.11 useTestEMVCards

```
- (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.

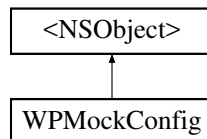
The documentation for this class was generated from the following file:

- WPCConfig.h

5.13 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](#)

5.13.1 Detailed Description

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

5.13.2 Property Documentation

5.13.2.1 batteryLevelError

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

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

5.13.2.2 cardReadFailure

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

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

5.13.2.3 cardReadTimeOut

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

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

5.13.2.4 cardTokenizationFailure

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

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

5.13.2.5 EMVAuthFailure

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

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

5.13.2.6 mockCardReaderIsDetected

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

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

5.13.2.7 mockPaymentMethod

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

The payment method to mock. Defaults to kWPPaymentMethodSwipe.

5.13.2.8 multipleEMVApplication

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

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

5.13.2.9 useMockCardReader

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

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

5.13.2.10 useMockWepayClient

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

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

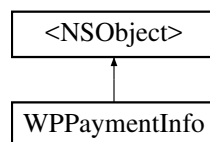
The documentation for this class was generated from the following file:

- WPMockConfig.h

5.14 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](#)
- WPAAddress * [billingAddress](#)
- WPAAddress * [shippingAddress](#)
- id [paymentMethod](#)
- id [swiperInfo](#)
- id [manualInfo](#)
- id [emvInfo](#)

5.14.1 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.

5.14.2 Method Documentation

5.14.2.1 addEmail:()

```
- (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

<i>email</i>	the email address to be added
--------------	-------------------------------

5.14.2.2 initWithFirstName:lastName:email:billingAddress:shippingAddress:cardNumber:cvv:expMonth:expYear:virtualTerminal:()

```
- (instancetype) initWithFirstName:
    (NSString *) firstName
    lastName:(NSString *) lastName
    email:(NSString *) email
    billingAddress:(WPAAddress *) billingAddress
    shippingAddress:(WPAAddress *) shippingAddress
    cardNumber:(NSString *) cardNumber
    cvv:(NSString *) cvv
```

```
expMonth:(NSString *) expMonth
expYear:(NSString *) expYear
virtualTerminal:(BOOL) virtualTerminal
```

Initializes a [WPPaymentInfo](#) instance of type kWPPaymentMethodManual.

Parameters

<i>firstName</i>	First name of the payer.
<i>lastName</i>	Last name of the payer.
<i>email</i>	Email address of the payer.
<i>billingAddress</i>	Billing address.
<i>shippingAddress</i>	Shipping address.
<i>cardNumber</i>	The card number.
<i>cvv</i>	The cvv code.
<i>expMonth</i>	The 2-digit expiration month on the credit card.
<i>expYear</i>	The 4-digit expiration year on the credit card.
<i>virtualTerminal</i>	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.

5.14.3 Property Documentation

5.14.3.1 `billingAddress`

```
- (WPAAddress*) billingAddress [read], [nonatomic], [strong]
```

Billing address.

5.14.3.2 `email`

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

Email address of the payer.

5.14.3.3 `emvInfo`

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

Additional info obtained by using the EMV payment method.

5.14.3.4 firstName

- (NSString*) firstName [read], [nonatomic], [strong]

First name of the payer.

5.14.3.5 isVirtualTerminal

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

Determines if the card was obtained in virtual terminal mode.

5.14.3.6 lastName

- (NSString*) lastName [read], [nonatomic], [strong]

Last name of the payer.

5.14.3.7 manualInfo

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

Additional info obtained by using the Manual payment method.

5.14.3.8 paymentDescription

- (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.

5.14.3.9 paymentMethod

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

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

5.14.3.10 shippingAddress

- (WPAAddress*) shippingAddress [read], [nonatomic], [strong]

Shipping address.

5.14.3.11 swiperInfo

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

Additional info obtained by using the Swipe payment method.

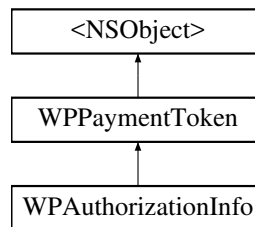
The documentation for this class was generated from the following file:

- WPPaymentInfo.h

5.15 WPPaymentToken Class Reference

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

Inheritance diagram for WPPaymentToken:



Instance Methods

- (instancetype) - initWithId:

Properties

- NSString * tokenId

5.15.1 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.

5.15.2 Method Documentation

5.15.2.1 initWithId:()

```
- (instancetype) initWithId:
    (NSString *) tokenId
```

Initializes a [WPPaymentToken](#) with the Id provided.

Parameters

<i>token↔ id</i>	The Id of the token.
----------------------	----------------------

Returns

A [WPPaymentToken](#) object initialized with the Id provided.

5.15.3 Property Documentation

5.15.3.1 tokenId

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

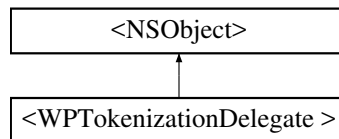
The token's id.

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

- WPPaymentToken.h

5.16 <WPTokenizationDelegate> Protocol Reference

Inheritance diagram for <WPTokenizationDelegate>:



Instance Methods

- (void) - [paymentInfo:didTokenize:](#)
- (void) - [paymentInfo:didFailTokenization:](#)
- (void) - [insertPayerEmailWithCompletion:](#)

5.16.1 Method Documentation

5.16.1.1 insertPayerEmailWithCompletion:()

```
- (void WPTokenizationDelegate) 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

<i>completion</i>	The block to be executed with the payer's email address.
<i>email</i>	The payer's email address.

5.16.1.2 `paymentInfo:didFailTokenization:()`

```
- (void WPTokenizationDelegate) paymentInfo:
    (WPPaymentInfo *) paymentInfo
    didFailTokenization:(NSError *) error
```

Called when a tokenization call fails.

Parameters

<i>paymentInfo</i>	The payment info that failed tokenization.
<i>error</i>	The error which caused the failure.

5.16.1.3 `paymentInfo:didTokenize:()`

```
- (void WPTokenizationDelegate) paymentInfo:
    (WPPaymentInfo *) paymentInfo
    didTokenize:(WPPaymentToken *) paymentToken
```

Called when a tokenization call succeeds.

Parameters

<i>paymentInfo</i>	The payment info that was tokenized.
<i>paymentToken</i>	The payment token representing the payment info.

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

- WePay.h

5.17 <WPTokenizationDelegate> Protocol Reference

```
#include <WePay.h>
```

5.17.1 Detailed Description

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

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

- WePay.h

6 File Documentation

6.1 WPErrors.h File Reference

[WPErrors.h](#) serves as documentation for all errors surfaced by the [WePay](#) iOS SDK.

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

Macros

- `#define WPUnexpectedErrorMessage` `NSStringFromTable(@"There was an unexpected error.", @"WePay", @"There was an unexpected error.");`
- `#define WPNoDataReturnedErrorMessage` `NSStringFromTable(@"There was no data returned.", @"WePay", @"There was no data returned.");`
- `#define WPCardReaderGeneralErrorMessage` `NSStringFromTable(@"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` `NSStringFromTable(@"Failed to initialize card reader.", @"WePay", @"Failed to initialize card reader.");`
- `#define WPCardReaderTimeoutErrorMessage` `NSStringFromTable(@"Card reader timed out.", @"WePay", @"Card reader timed out.");`
- `#define WPSignatureInvalidImageErrorMessage` `NSStringFromTable(@"Invalid signature image provided.", @"WePay", @"Invalid signature image provided.");`
- `#define WPNameNotFoundErrorMessage` `NSStringFromTable(@"Name not found.", @"WePay", @"Name not found.");`
- `#define WPInvalidCardDataErrorMessage` `NSStringFromTable(@"Invalid card data.", @"WePay", @"Invalid card data.");`
- `#define WPCardNotSupportedErrorMessage` `NSStringFromTable(@"This card is not supported.", @"WePay", @"This card is not supported.");`
- `#define WPInvalidApplicationIdErrorMessage` `NSStringFromTable(@"Invalid application ID selected.", @"WePay", @"Invalid application ID selected.");`
- `#define WPDeclinedByCardErrorMessage` `NSStringFromTable(@"The transaction was declined by the card.", @"WePay", @"The transaction was declined by the card.");`
- `#define WPCardBlockedErrorMessage` `NSStringFromTable(@"This card has been blocked.", @"WePay", @"This card has been blocked.");`
- `#define WPDeclinedByIssuerErrorMessage` `NSStringFromTable(@"The transaction was declined by the issuer bank.", @"WePay", @"The transaction was declined by the issuer bank.");`
- `#define WPIssuerUnreachableErrorMessage` `NSStringFromTable(@"The issuing bank could not be reached.", @"WePay", @"The issuing bank could not be reached.");`

- #define `WPInvalidAuthInfoErrorMessage` `NSStringFromTable(@"The provided auth info is invalid.", @"WePay", @"The provided auth info is invalid.");`
- #define `WPAuthInfoNotProvidedErrorMessage` `NSStringFromTable(@"Auth info was not provided.", @"WePay", @"Auth info was not provided.");`
- #define `WPPaymentMethodCannotBeTokenizedErrorMessage` `NSStringFromTable(@"This payment method cannot be tokenized.", @"WePay", @"This payment method cannot be tokenized.");`
- #define `WPFailedToGetBatteryLevelErrorMessage` `NSStringFromTable(@"Battery level could not be determined.", @"WePay", @"Battery level could not be determined.");`
- #define `WPCardReaderNotConnectedErrorMessage` `NSStringFromTable(@"Card reader is not connected.", @"WePay", @"Card reader is not connected.");`
- #define `WPCardReaderModelNotSupportedErrorMessage` `NSStringFromTable(@"This card reader model is not supported.", @"WePay", @"This card reader model is not supported.");`
- #define `WPErrInvalidTransactionAmountErrorMessage` `NSStringFromTable(@"The provided transaction amount is invalid.", @"WePay", @"The provided transaction amount is invalid.");`
- #define `WPErrInvalidTransactionCurrencyCodeErrorMessage` `NSStringFromTable(@"The provided currency code is invalid.", @"WePay", @"The provided currency code is invalid.");`
- #define `WPErrInvalidTransactionAccountIDErrorMessage` `NSStringFromTable(@"The provided account ID is invalid.", @"WePay", @"The provided account ID is invalid.");`
- #define `WPErrInvalidCardReaderSelectionErrorMessage` `NSStringFromTable(@"Card reader selection is invalid.", @"WePay", @"Card reader selection is invalid.");`
- #define `WPErrCardReaderBatteryTooLowErrorMessage` `NSStringFromTable(@"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 `WPErrCardReaderUnableToConnectErrorMessage` `NSStringFromTable(@"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 `WPErrCode` {
`WPErrUnknown` = -10000, `WPErrNoDataReturned` = -10015, `WPErrCardReaderGeneralError` = -10016,
`WPErrCardReaderInitialization` = -10017,
`WPErrCardReaderTimeout` = -10018, `WPErrCardReaderStatusError` = -10019, `WPErrInvalidSignatureImage` = -10020, `WPErrNameNotFound` = -10021,
`WPErrInvalidCardData` = -10022, `WPErrCardNotSupported` = -10023, `WPErrEMVTransactionError` = -10024, `WPErrInvalidApplicationId` = -10025,
`WPErrDeclinedByCard` = -10026, `WPErrCardBlocked` = -10027, `WPErrDeclinedByIssuer` = -10028, `WPErrIssuerUnreachable` = -10029,
`WPErrInvalidAuthInfo` = -10030, `WPErrAuthInfoNotProvided` = -10031, `WPErrPaymentMethodCannotBeTokenized` = -10032, `WPErrFailedToGetBatteryLevel` = -10033,
`WPErrCardReaderNotConnected` = -10034, `WPErrCardReaderModelNotSupported` = -10035, `WPErrInvalidTransactionAmount` = -10036, `WPErrInvalidTransactionCurrencyCode` = -10037,
`WPErrInvalidTransactionAccountID` = -10038, `WPErrInvalidCardReaderSelection` = -10039, `WPErrCardReaderBatteryTooLow` = -10040, `WPErrCardReaderUnableToConnect` = -10041 }

Variables

- FOUNDATION_EXPORT NSString *const `kWPErrAPIDomain`
- FOUNDATION_EXPORT NSString *const `kWPErrSDKDomain`
- FOUNDATION_EXPORT NSString *const `kWPErrCategoryKey`
- FOUNDATION_EXPORT NSString *const `kWPErrCategoryNone`
- FOUNDATION_EXPORT NSString *const `kWPErrCategoryCardReader`
- FOUNDATION_EXPORT NSString *const `kWPErrCategoryCardSDK`
- enum `WPErrCode` **WPErrCode**

6.1.1 Detailed Description

[WPErrors.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 kWPErrorsCategoryKey and NSLocalizedDescriptionKey

The [WePay](#) iOS SDK can return errors in various error domains:

- [WePay](#) server API errors are in the kWPErrorsAPIDomain
- Errors generated by the SDK itself are in the kWPErrorsSDKDomain
- System errors generated by iOS are passed through as-is, for example in the NSErrorDomain

See the [WPErrorsCode](#) section for more details about error codes.

6.1.2 Macro Definition Documentation

6.1.2.1 WPAuthInfoNotProvidedErrorMessage

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

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

6.1.2.2 WPCardBlockedErrorMessage

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

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

6.1.2.3 WPCardNotSupportedErrorMessage

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

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

6.1.2.4 WPCardReaderGeneralErrorMessage

```
#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 `WPErrCardReaderGeneralError`, that can be retrieved by calling `[error localizedDescription]`.

6.1.2.5 WPCardReaderInitializationErrorMessage

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

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

6.1.2.6 WPCardReaderModelNotSupportedErrorMessage

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

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

6.1.2.7 WPCardReaderNotConnectedErrorMessage

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

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

6.1.2.8 WPCardReaderTimeoutErrorMessage

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

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

6.1.2.9 WPDeclinedByCardErrorMessage

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

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

6.1.2.10 WPDeclinedByIssuerErrorMessage

```
#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 `WPErrorsDeclinedByIssuer`, that can be retrieved by calling `[error localizedDescription]`.

6.1.2.11 WPErrorsCardReaderBatteryTooLowErrorMessage

```
#define WPErrorsCardReaderBatteryTooLowErrorMessage 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 `WPErrorsCardReaderBatteryTooLow`, that can be retrieved by calling `[error localizedDescription]`.

6.1.2.12 WPErrorsCardReaderUnableToConnectErrorMessage

```
#define WPErrorsCardReaderUnableToConnectErrorMessage 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 `WPErrorsCardReaderUnsupportedOrBatteryTooLow`, that can be retrieved by calling `[error localizedDescription]`.

6.1.2.13 WPErrorsInvalidCardReaderSelectionErrorMessage

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

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

6.1.2.14 WPErrorsInvalidTransactionAccountIDErrorMessage

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

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

6.1.2.15 WPErrorsInvalidTransactionAmountErrorMessage

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

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

6.1.2.16 WPErrorInvalidTransactionCurrencyCodeErrorMessage

```
#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 WPFailedToGetBatteryLevelErrorMessage

```
#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 WPInvalidApplicationIdErrorMessage

```
#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 localizedDescription].

6.1.2.19 WPInvalidAuthInfoErrorMessage

```
#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 localizedDescription].

6.1.2.20 WPInvalidCardDataErrorMessage

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

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

6.1.2.21 WPIssuerUnreachableErrorMessage

```
#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 localizedDescription].

6.1.2.22 WPNameNotFoundErrorMessage

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

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

6.1.2.23 WPNoDataReturnedErrorMessage

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

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

6.1.2.24 WPPaymentMethodCannotBeTokenizedErrorMessage

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

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

6.1.2.25 WPSignatureInvalidImageErrorMessage

```
#define WPSignatureInvalidImageErrorMessage NSLocalizedStringFromTable(@"Invalid signature image provided.", @"WePay", @"Invalid signature image provided.");
```

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

6.1.2.26 WPUnexpectedErrorMessage

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

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

6.1.3 Enumeration Type Documentation

6.1.3.1 WPErrorsCode

```
enum WPErrorsCode
```

Error codes for `NSError`s surfaced by the [WePay](#) iOS SDK in the `kWPErrorsSDKDomain`. For a full list of error codes in the `kWPErrorsAPIDomain`, visit <https://www.wepay.com/developer/reference/errors>

Enumerator

WPErrUnknown	-10000 Unknown error.
WPErrNoDataReturned	-10015 No data returned by the API call.
WPErrCardReaderGeneralError	-10016 General error reported by the card reader - usually due to a bad swipe.
WPErrCardReaderInitialization	-10017 Error while initializing the card reader.
WPErrCardReaderTimeout	-10018 Timeout occurred while waiting for card.
WPErrCardReaderStatusError	-10019 Special error reported by card reader - very rare.
WPErrInvalidSignatureImage	-10020 Invalid signature image.
WPErrNameNotFound	-10021 Name not found.
WPErrInvalidCardData	-10022 Invalid card data.
WPErrCardNotSupported	-10023 Card not supported.
WPErrEMVTransactionError	-10024 EMV transaction error.
WPErrInvalidApplicationId	-10025 Invalid application ID.
WPErrDeclinedByCard	-10026 Declined by card.
WPErrCardBlocked	-10027 Card blocked.
WPErrDeclinedByIssuer	-10028 Declined by issuer.
WPErrIssuerUnreachable	-10029 Issuer unreachable.
WPErrInvalidAuthInfo	-10030 Invalid auth info.
WPErrAuthInfoNotProvided	-10031 Auth info not provided.
WPErrPaymentMethodCannotBeTokenized	-10032 Payment method cannot be tokenized.
WPErrFailedToGetBatteryLevel	-10033 Failed to get battery level.
WPErrCardReaderNotConnected	-10034 Card reader not connected.
WPErrCardReaderModelNotSupported	-10035 Card reader model not supported.
WPErrInvalidTransactionAmount	-10036 Invalid transaction amount.
WPErrInvalidTransactionCurrencyCode	-10037 Invalid transaction currency code.
WPErrInvalidTransactionAccountID	-10038 Invalid transaction account id.
WPErrInvalidCardReaderSelection	-10039 Invalid card reader selection.
WPErrCardReaderBatteryTooLow	-10040 Card reader battery too low.
WPErrCardReaderUnableToConnect	-10041 Unable to connect to card reader.

6.1.4 Variable Documentation

6.1.4.1 kWPErrAPIDomain

```
FOUNDATION_EXPORT NSString* const kWPErrAPIDomain
```

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 [kWPErrAPIDomain](#), visit <https://www.wepay.com/developer/reference/errors>

6.1.4.2 kWPErrCategoryCardReader

```
FOUNDATION_EXPORT NSString* const kWPErrCategoryCardReader
```

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

6.1.4.3 kWPErrorsCategoryCardSDK

```
FOUNDATION_EXPORT NSString* const kWPErrorsCategoryCardSDK
```

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

6.1.4.4 kWPErrorsCategoryKey

```
FOUNDATION_EXPORT NSString* const kWPErrorsCategoryKey
```

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

6.1.4.5 kWPErrorsCategoryNone

```
FOUNDATION_EXPORT NSString* const kWPErrorsCategoryNone
```

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

6.1.4.6 kWPErrorsSDKDomain

```
FOUNDATION_EXPORT NSString* const kWPErrorsSDKDomain
```

The NSError domain of all errors returned by the [WePay](#) iOS SDK itself. For a full list of error codes in the [kWPErrorsSDKDomain](#), look at [WPErrorsCode](#).

Index

- <WPAuthorizationDelegate >, [22](#)
 - <WPAuthorizationDelegate>, [23](#)
 - <WPBatteryLevelDelegate >, [26](#)
 - <WPBatteryLevelDelegate>, [27](#)
 - <WPCardReaderDelegate >, [27](#)
 - <WPCardReaderDelegate>, [30](#)
 - <WPCheckoutDelegate >, [30](#)
 - <WPCheckoutDelegate>, [32](#)
 - <WPTokenizationDelegate >, [43](#)
 - <WPTokenizationDelegate>, [44](#)
 - addEmail:
 - WPPaymentInfo, [39](#)
 - address1
 - WPAAddress, [21](#)
 - address2
 - WPAAddress, [21](#)
 - amount
 - WPAuthorizationInfo, [25](#)
 - authorizeAmountWithCompletion:
 - WPCardReaderDelegate -p, [27](#)
 - batteryLevelError
 - WPMockConfig, [37](#)
 - billingAddress
 - WPPaymentInfo, [40](#)
 - callDelegateMethodsOnMainThread
 - WPConfig, [34](#)
 - cardReadFailure
 - WPMockConfig, [37](#)
 - cardReadTimeOut
 - WPMockConfig, [37](#)
 - cardReaderDidChangeStatus:
 - WPCardReaderDelegate -p, [28](#)
 - cardTokenizationFailure
 - WPMockConfig, [37](#)
 - city
 - WPAAddress, [21](#)
 - clientId
 - WPConfig, [34](#)
 - config
 - WePay, [18](#)
 - country
 - WPAAddress, [21](#)
 - currencyCode
 - WPAuthorizationInfo, [25](#)
 - didFailToGetBatteryLevelWithError:
 - WPBatteryLevelDelegate -p, [26](#)
 - didFailToReadPaymentInfoWithError:
 - WPCardReaderDelegate -p, [28](#)
 - didFailToStoreSignatureImage:forCheckoutId:withError:
 - WPCheckoutDelegate -p, [31](#)
 - didGetBatteryLevel:
 - WPBatteryLevelDelegate -p, [26](#)
 - didReadPaymentInfo:
 - WPCardReaderDelegate -p, [28](#)
 - didStoreSignature:forCheckoutId:
 - WPCheckoutDelegate -p, [32](#)
 - EMVAuthFailure
 - WPMockConfig, [37](#)
 - email
 - WPPaymentInfo, [40](#)
 - envInfo
 - WPPaymentInfo, [40](#)
 - environment
 - WPConfig, [35](#)
 - firstName
 - WPPaymentInfo, [40](#)
 - forgetRememberedCardReader
 - WePay, [15](#)
 - getCardReaderBatteryLevelWithCardReaderDelegate↵
 - :batteryLevelDelegate:
 - WePay, [15](#)
 - getRememberedCardReader
 - WePay, [16](#)
 - initWithAddress1:address2:city:region:postcode:country:
 - WPAAddress, [20](#)
 - initWithAddress1:address2:city:state:zip:
 - WPAAddress, [20](#)
 - initWithAmount:currencyCode:transactionToken:tokenId:
 - WPAuthorizationInfo, [24](#)
 - initWithClientId:environment:
 - WPConfig, [33](#)
 - initWithClientId:environment:useLocation:useTest↵
 - EMVCards:callDelegateMethodsOnMain↵
 - Thread:restartTransactionAfterSuccess↵
 - :restartTransactionAfterGeneralError:restart↵
 - TransactionAfterOtherErrors:stopCardReader↵
 - AfterOperation:logLevel:
 - WPConfig, [33](#)
- initWithConfig:
 - WePay, [16](#)
- initWithFirstName:lastName:email:billingAddress↵
 - :shippingAddress:cardNumber:cvv:expMonth↵
 - :expYear:virtualTerminal:
 - WPPaymentInfo, [39](#)
- initWithId:
 - WPAuthorizationInfo, [24](#)

- WPPaymentToken, 42
- initWithZip:
 - WPAAddress, 21
- insertPayerEmailWithCompletion:
 - WPTokenizationDelegate -p, 43
- isVirtualTerminal
 - WPPaymentInfo, 41
- kWPErrorAPIDomain
 - WPError.h, 52
- kWPErrorCategoryCardReader
 - WPError.h, 52
- kWPErrorCategoryCardSDK
 - WPError.h, 52
- kWPErrorCategoryKey
 - WPError.h, 53
- kWPErrorCategoryNone
 - WPError.h, 53
- kWPErrorSDKDomain
 - WPError.h, 53
- lastName
 - WPPaymentInfo, 41
- logLevel
 - WPConfig, 35
- manualInfo
 - WPPaymentInfo, 41
- mockCardReaderIsDetected
 - WPMockConfig, 37
- mockConfig
 - WPConfig, 35
- mockPaymentMethod
 - WPMockConfig, 37
- multipleEMVApplication
 - WPMockConfig, 37
- paymentDescription
 - WPPaymentInfo, 41
- paymentInfo:didAuthorize:
 - WPAuthorizationDelegate -p, 22
- paymentInfo:didFailAuthorization:
 - WPAuthorizationDelegate -p, 23
- paymentInfo:didFailTokenization:
 - WPTokenizationDelegate -p, 44
- paymentInfo:didTokenize:
 - WPTokenizationDelegate -p, 44
- paymentMethod
 - WPPaymentInfo, 41
- postcode
 - WPAAddress, 21
- region
 - WPAAddress, 22
- restartTransactionAfterGeneralError
 - WPConfig, 35
- restartTransactionAfterOtherErrors
 - WPConfig, 35
- restartTransactionAfterSuccess
 - WPConfig, 35
- selectCardReader:completion:
 - WPCardReaderDelegate -p, 29
- selectEMVApplication:completion:
 - WPCardReaderDelegate -p, 29
- shippingAddress
 - WPPaymentInfo, 41
- shouldResetCardReaderWithCompletion:
 - WPCardReaderDelegate -p, 29
- startTransactionForReadingWithCardReaderDelegate:
 - WePay, 16
- startTransactionForTokenizingWithCardReaderDelegate↵:tokenizationDelegate:authorizationDelegate:
 - WePay, 17
- state
 - WPAAddress, 22
- stopCardReader
 - WePay, 18
- stopCardReaderAfterOperation
 - WPConfig, 35
- storeSignatureImage:forCheckoutId:checkoutDelegate:
 - WePay, 18
- swiperInfo
 - WPPaymentInfo, 41
- tokenId
 - WPAuthorizationInfo, 25
 - WPPaymentToken, 43
- tokenizePaymentInfo:tokenizationDelegate:
 - WePay, 18
- transactionToken
 - WPAuthorizationInfo, 25
- useLocation
 - WPConfig, 35
- useMockCardReader
 - WPMockConfig, 38
- useMockWepayClient
 - WPMockConfig, 38
- useTestEMVCards
 - WPConfig, 36
- WPAAddress, 19
 - address1, 21
 - address2, 21
 - city, 21
 - country, 21
 - initWithAddress1:address2:city:region:postcode↵:country:, 20
 - initWithAddress1:address2:city:state:zip:, 20

- initWithZip:, 21
- postcode, 21
- region, 22
- state, 22
- zip, 22
- WPAuthInfoNotProvidedErrorMessage
 - WPErr.h, 47
- WPAuthorizationDelegate -p
 - paymentInfo:didAuthorize:, 22
 - paymentInfo:didFailAuthorization:, 23
- WPAuthorizationInfo, 23
 - amount, 25
 - currencyCode, 25
 - initWithAmount:currencyCode:transactionToken↵:tokenId:, 24
 - initWithId:, 24
 - tokenId, 25
 - transactionToken, 25
- WPBatteryLevelDelegate -p
 - didFailToGetBatteryLevelWithError:, 26
 - didGetBatteryLevel:, 26
- WPCardBlockedErrorMessage
 - WPErr.h, 47
- WPCardNotSupportedErrorMessage
 - WPErr.h, 47
- WPCardReaderDelegate -p
 - authorizeAmountWithCompletion:, 27
 - cardReaderDidChangeStatus:, 28
 - didFailToReadPaymentInfoWithError:, 28
 - didReadPaymentInfo:, 28
 - selectCardReader:completion:, 29
 - selectEMVApplication:completion:, 29
 - shouldResetCardReaderWithCompletion:, 29
- WPCardReaderGeneralErrorMessage
 - WPErr.h, 47
- WPCardReaderInitializationErrorMessage
 - WPErr.h, 48
- WPCardReaderModelNotSupportedErrorMessage
 - WPErr.h, 48
- WPCardReaderNotConnectedErrorMessage
 - WPErr.h, 48
- WPCardReaderTimeoutErrorMessage
 - WPErr.h, 48
- WPCheckoutDelegate -p
 - didFailToStoreSignatureImage:forCheckoutId:with↵Error:, 31
 - didStoreSignature:forCheckoutId:, 32
- WPConfig, 32
 - callDelegateMethodsOnMainThread, 34
 - clientId, 34
 - environment, 35
 - initWithClientId:environment:, 33
 - initWithClientId:environment:useLocation:use↵TestEMVCards:callDelegateMethodsOn↵
 - MainThread:restartTransactionAfterSuccess↵:restartTransactionAfterGeneralError:restart↵TransactionAfterOtherErrors:stopCardReader↵AfterOperation:logLevel:, 33
 - logLevel, 35
 - mockConfig, 35
 - restartTransactionAfterGeneralError, 35
 - restartTransactionAfterOtherErrors, 35
 - restartTransactionAfterSuccess, 35
 - stopCardReaderAfterOperation, 35
 - useLocation, 35
 - useTestEMVCards, 36
- WPDeclinedByCardErrorMessage
 - WPErr.h, 48
- WPDeclinedByIssuerErrorMessage
 - WPErr.h, 48
- WPErr.h, 45
 - kWPErrAPIDomain, 52
 - kWPErrCategoryCardReader, 52
 - kWPErrCategoryCardSDK, 52
 - kWPErrCategoryKey, 53
 - kWPErrCategoryNone, 53
 - kWPErrSDKDomain, 53
 - WPAuthInfoNotProvidedErrorMessage, 47
 - WPCardBlockedErrorMessage, 47
 - WPCardNotSupportedErrorMessage, 47
 - WPCardReaderGeneralErrorMessage, 47
 - WPCardReaderInitializationErrorMessage, 48
 - WPCardReaderModelNotSupportedErrorMessage, 48
 - WPCardReaderNotConnectedErrorMessage, 48
 - WPCardReaderTimeoutErrorMessage, 48
 - WPDeclinedByCardErrorMessage, 48
 - WPDeclinedByIssuerErrorMessage, 48
 - WPErrCardReaderBatteryTooLowErrorMessage, 49
 - WPErrCardReaderUnableToConnectError↵Message, 49
 - WPErrCode, 51
 - WPErrInvalidCardReaderSelectionErrorMessage, 49
 - WPErrInvalidTransactionAccountIDErrorMessage, 49
 - WPErrInvalidTransactionAmountErrorMessage, 49
 - WPErrInvalidTransactionCurrencyCodeError↵Message, 49
 - WPFailedToGetBatteryLevelErrorMessage, 50
 - WPInvalidApplicationIdErrorMessage, 50
 - WPInvalidAuthInfoErrorMessage, 50
 - WPInvalidCardDataErrorMessage, 50
 - WPIssuerUnreachableErrorMessage, 50
 - WPNameNotFoundErrorMessage, 50
 - WPNoDataReturnedErrorMessage, 51

- WPPaymentMethodCannotBeTokenizedError↔
Message, 51
- WPSignatureInvalidImageErrorMessage, 51
- WPUUnexpectedErrorMessage, 51
- WPErrCardReaderBatteryTooLowErrorMessage
WPErr.h, 49
- WPErrCardReaderUnableToConnectErrorMessage
WPErr.h, 49
- WPErrCode
WPErr.h, 51
- WPErrInvalidCardReaderSelectionErrorMessage
WPErr.h, 49
- WPErrInvalidTransactionAccountIDErrorMessage
WPErr.h, 49
- WPErrInvalidTransactionAmountErrorMessage
WPErr.h, 49
- WPErrInvalidTransactionCurrencyCodeErrorMessage
WPErr.h, 49
- WPFailedToGetBatteryLevelErrorMessage
WPErr.h, 50
- WPIInvalidApplicationIDErrorMessage
WPErr.h, 50
- WPIInvalidAuthInfoErrorMessage
WPErr.h, 50
- WPIInvalidCardDataErrorMessage
WPErr.h, 50
- WPIssuerUnreachableErrorMessage
WPErr.h, 50
- WPMockConfig, 36
 - batteryLevelError, 37
 - cardReadFailure, 37
 - cardReadTimeOut, 37
 - cardTokenizationFailure, 37
 - EMVAuthFailure, 37
 - mockCardReaderIsDetected, 37
 - mockPaymentMethod, 37
 - multipleEMVApplication, 37
 - useMockCardReader, 38
 - useMockWepayClient, 38
- WPNameNotFoundErrorMessage
WPErr.h, 50
- WPNoDataReturnedErrorMessage
WPErr.h, 51
- WPPaymentInfo, 38
 - addEmail, 39
 - billingAddress, 40
 - email, 40
 - emvInfo, 40
 - firstName, 40
 - initWithFirstName:lastName:email:billingAddress↔
:shippingAddress:cardNumber:cvv:expMonth↔
:expYear:virtualTerminal, 39
 - isVirtualTerminal, 41
 - lastName, 41
 - manualInfo, 41
 - paymentDescription, 41
 - paymentMethod, 41
 - shippingAddress, 41
 - swiperInfo, 41
- WPPaymentMethodCannotBeTokenizedErrorMessage
WPErr.h, 51
- WPPaymentToken, 42
 - initWithId, 42
 - tokenId, 43
- WPSignatureInvalidImageErrorMessage
WPErr.h, 51
- WPTokenizationDelegate -p
 - insertPayerEmailWithCompletion, 43
 - paymentInfo:didFailTokenization, 44
 - paymentInfo:didTokenize, 44
- WPUUnexpectedErrorMessage
WPErr.h, 51
- WePay, 14
 - config, 18
 - forgetRememberedCardReader, 15
 - getCardReaderBatteryLevelWithCardReader↔
Delegate:batteryLevelDelegate, 15
 - getRememberedCardReader, 16
 - initWithConfig, 16
 - startTransactionForReadingWithCardReader↔
Delegate, 16
 - startTransactionForTokenizingWithCardReader↔
Delegate:tokenizationDelegate:authorization↔
Delegate, 17
 - stopCardReader, 18
 - storeSignatureImage:forCheckoutId:checkout↔
Delegate, 18
 - tokenizePaymentInfo:tokenizationDelegate, 18
- zip
 - WPAAddress, 22