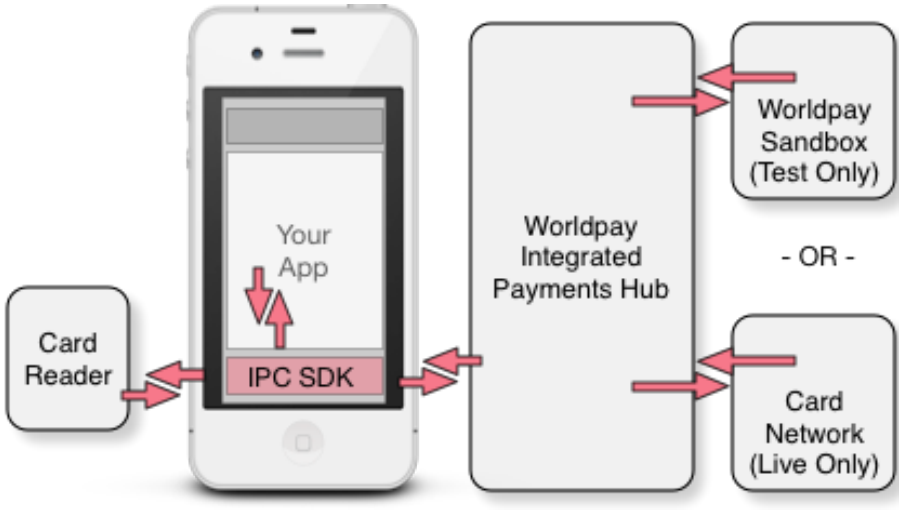
Introduction

The Worldpay Total Integrated Payment Client Software Development Kit (IPC SDK) is a wonderful way for you to develop rich mobile applications in the native language of the mobile device. We currently have SDKs for iOS, Android, and Windows. Besides allowing you to develop native applications, there are two other compelling reasons to choose this platform:

- We do all of the device management of your card reader. No need to worry about device drivers. No Bluetooth management functions. Nothing. We take care of those and report back to you when something of interest happens.
- You do not have to undergo EMV certification testing, because we protect you from the card data. This would otherwise be a costly and time-consuming activity.

The SDKs provide you with the glue necessary to talk with a card reader (if you use one) and the code to make the remote calls to the Worldpay Integrated Payments Hub.



Note that while one of the advantages to using the IPC SDK is the management of the card reader, you can still use this library for card not present applications or any other app that wants access to our vault, tokenization, and customer management features. Think you might one day move from card not present to card present? Develop against the app today and rest comfortable that your app will be future-proofed when you are ready to make the change – same code base.

The IPC SDK installs alongside your software - adding transaction processing to your applications. It facilitates all transactional communication between the Worldpay Integrated Payments Hub (IPH) and approved hardware devices to isolate payment data and keeps it separate from your software application.

Requirements

Platform

iOS 8 and later

Supported Devices

Miura M010 Shuttle

Overview of the Software Development Kit API

Your starting point for all of the operations available in the IPC SDK is the WorldPayAPI object. This object is a singleton accessible by the `instance` method on the `WorldPayAPI` class.

Securely Acquiring Card Information

The IPC SDK manages all interaction with the attached card reader, and then uses the card account information to perform transactions through the Worldpay Total Integrated Payment Hub. The IPC SDK also provides a way for you to manually enter card information for those times when the card is not present for the transaction.

The IPC SDK supports the following methods to capture card information:

Card Information Capture Methods	
Address the Attached Device	Gets a reference to the attached device and allows you to set up handlers for device events. See: `WorldPayAPI.swiperWithDelegate()`
Magnetic Stripe Reader	Captures the card information from the magnetic stripe reader on the attached device. See: `WPYSwiper.beginEMVTransaction()`
EMV Contact	Obtains card information from a chip card inserted in the EMV reader of the attached device. See: `WPYSwiper.beginEMVTransaction()`
Manual Entry	Presents a form where the customer can manually enter the card number, expiration date and postal code, and then calls the Integrated Payment Hub to perform the specified payment transaction. Returns the result of the payment transaction to the associated delegate. See: `WPYManualTenderEntryViewController`

Making Payments

The following table shows the transactions you can use to manage payments. The IPC SDK takes care of the network interactions with the Integrated Payment Hub.

Payment Transactions	
Authorize	This call authorizes a transaction but does not capture the transaction for settlement. In a card-present environment, this option is most commonly used in service industry transactions where an initial amount is authorized prior to a gratuity being added. If the Authorization Only call is used, a subsequent Prior Auth Capture call must be made to capture the transaction for settlement. See: `WorldPayAPI.paymentAuthorize()`

Payment Transactions	
Prior Auth Capture	This call allows a previously authorized transaction to be captured for settlement. In a card-present environment, Prior Auth Capture is commonly used when the transaction amount must be modified during capture, e.g., in the service industry when a gratuity is added to the original amount. See: `WorldPayAPI.paymentCapture()`
Charge	This call authorizes the transaction and, if successful, captures it. This is the most common call for card-present transactions. (For some transactions, it may be necessary to separate these steps, e.g., in the service industry example described above. In such cases, separate calls to Authorize Only and Prior Auth Capture should be used instead of Charge.) See: `WorldPayAPI.paymentCharge()`
Refund	The Refund method must be linked to a settled transaction. This is done by specifying the transactionId from the original Authorization or Charge as part of the request. By default, this method refunds the FULL amount of the transaction. However, you can perform a partial refund by passing a specific amount. If a refund is attempted on a transaction that has not yet settled, the Integrated Payment Hub will automatically run a Void on the transaction. The transactionType in this case will switch to Void. See: `WorldPayAPI.paymentRefund()`
Void	A void may be applied against any charge that has not yet settled; it effectively undoes the original authorization or charge as if it had never taken place. See: `WorldPayAPI.paymentVoid()`
Credit	A credit is a payment from a merchant account to a credit card or checking account that does not have to be linked to a previous transaction. To use the Credit method, the merchant must be specifically enabled for it on the Worldpay Total platform. This is not recommended due to the risk of misuse. See: `WorldPayAPI.paymentCredit()`

Controlling Settlement with Batches

All transactions that have been authorized and captured are associated with a batch. A batch is considered open until it is settled, which completes the captured transactions and allows funds to be transferred from the customers to the merchant. While open, transactions can still be altered (i.e. voids, adding a tip). Once a batch is settled, it is considered closed; no alterations can be made to transactions in a closed batch.

In some cases, your account may be configured to automatically close out all open transactions and start a new batch at a particular time every night. However, you can close a batch at any time

using the Close method. The API also provides reporting methods that allow you to retrieve details for both open and closed batches for a particular merchant.
All batches are automatically closed at 4:00AM ET.

Managing Batches	
Close a Batch	Closing the current open batch settles all captured transactions in the batch See: `WorldPayAPI.closeCurrentBatch()`
Retrieving the Current Batch ID	Retrieve the ID for the current batch See: `WorldPayAPI.getCurrentBatch()`
Retrieving Transactions in the a Batch	Calling this method retrieves the transactions in the specified batch. It returns an array of the transactions in the open batch, along with the full details of each as returned during the original authorization. See: `WorldPayAPI.getTransactionsInBatch()`

Setting up Your Project in Xcode

Step 1 – Get the Lastest IPC SDK

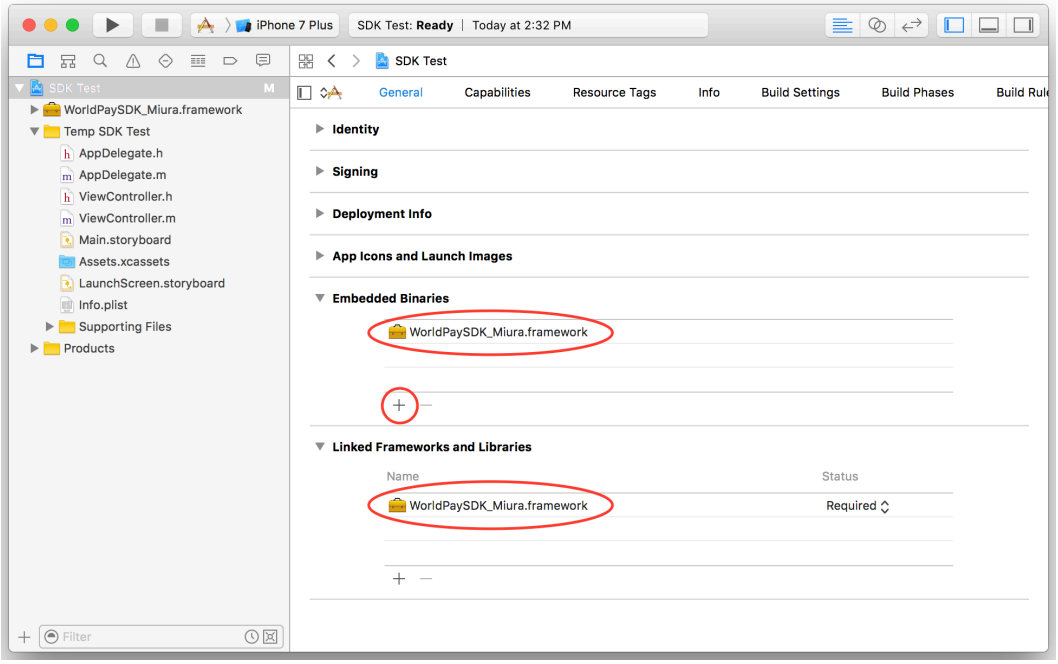
Download the latest IPC SDK framework for iOS from GitHub at:

http://www.github.com/worldpayus/ipc_sdk_ios/frameworks/WorldPaySDK_Miura.framework

Step 2 - Add the framework to your Xcode project

1. Select the "General" tab in your project's settings
2. Press the "+" button under the "Embedded Binaries" section
3. Press the "Add Other …" button on the "Choose items to add dialog"
4. Navigate to the `WorldPaySDK_Miura.framework` file you downloaded in the prior step and press "Open"

This will copy the file into your project, add the embedded binary and register the framework in your project. When you are done, the configuration should look like this:

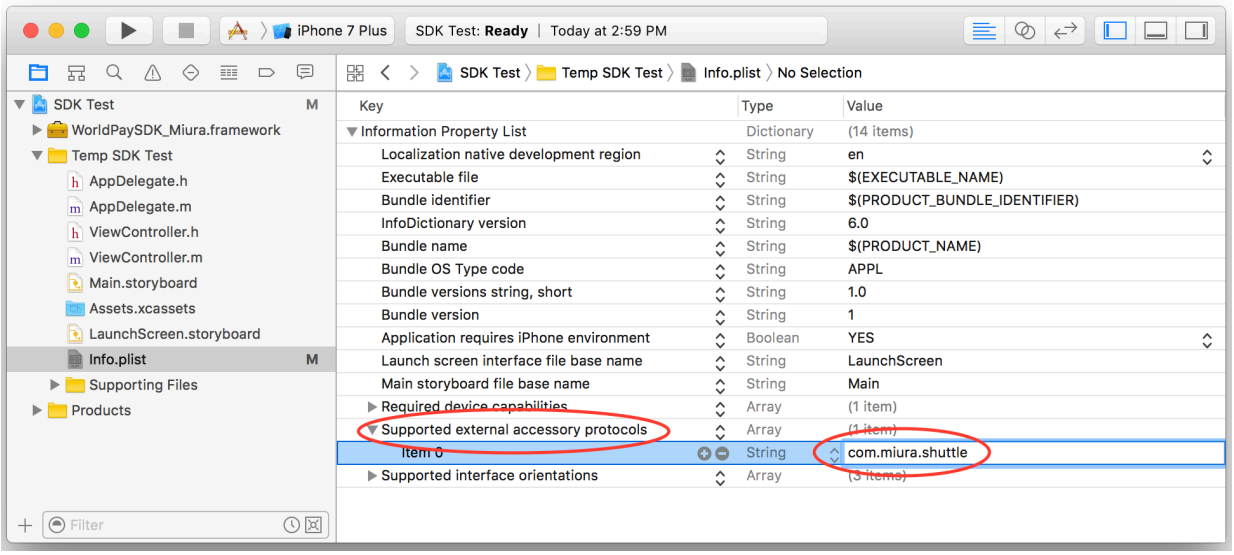


Step 3 – Add Your Device to Supported Protocols

Add the Miura device to the supported protocols in your Info.plist file

1. Navigate to the Info.plist file in your project
2. Left-click and add a "Supported external accessory protocols" row if it doesn't already exist
3. Add "com.miura.shuttle" to the list of protocols

When you are done, the configuration should look like this:



Your Application

Once the framework installed and the project has been configured, you can begin to code your application. You can use the features of the IPC SDK in a variety of ways to meet your payment needs.

To get you started here is a basic application flow:

- Obtain an authorization token
- Implement swiper delegate methods
- Get card information and process the transaction
- Handle the transaction response

Asynchronous Responses

Communication over the network to the Integrated Payment Hub happens asynchronously in a background thread. When you initiate a payment transaction, control returns immediately to your application. The IPC SDK manages the background thread and waits for the response from the Integrated Payment Hub.

The IPC SDK notifies your application about the response by calling delegate methods that you implement. Note that your delegate's methods are called on the application's main thread so you can safely update UI components.

Step 1 – Obtain an Authorization Token

Your application has to authenticate itself with the Integrated Payment Hub before you can make payment transactions. The token is cached and used for subsequent calls. This is called using `WorldPaySDK.generateAuthToken()` method which takes an `WPYAuthTokenRequest` object as input. Note that this authentication token represents the session with the Integrated payments Hub. It is not the same thing as a tokenized payment method. You will need to provide your `secureNetId` and `secureKey` (provided with your sandbox account) as illustrated below.

Setting a terminal ID and/or terminal vendor with the auth token causes that TID to populate automatically on all payments processed through the terminal hardware. If a TID is not set, and not populated prior to making a payment request, the terminal serial number will be used as the TID

```
#import <WorldPaySDK_Miura/WorldPaySDK.h>

WPYAuthTokenRequest *authTokenRequest = [[WPYAuthTokenRequest alloc] init];

authTokenRequest.secureNetId = @"0000000";
authTokenRequest.secureNetKey = @"xxxxxxxxxxx";
authTokenRequest.applicationId = @"applicationId";
authTokenRequest.terminalId = @"123";
authTokenRequest.terminalVendor = @"1234";

[[WorldpayAPI instance] generateAuthToken:authTokenRequest withCompletion:^(NSString *result, NSError *error) {
    if(!result || error) {
        NSLog(@"Error generating AUTH Token: %@", error);
    }
}];
```

The IPC SDK calls the `withCompletion` block[1] that you provide when the response is returned from the Integrated Payment Hub.

[1] Blocks in Objective-C are nameless functions that can be passed as arguments. They are similar to lambdas or closures in other languages. Blocks are declared starting with a caret (^) symbol. See the following Apple documentation for more information:

https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Blocks/Articles/00_Introduction.html

Step 2 – Implement Mandatory Swiper Delegate Methods

`WPYSwiper` is a singleton class that represents the attached card reader. Your application is notified about actions taken by the card reader through methods on a `WPYSwiperDelegate` that you provide. The delegate defines a number of callback methods, some of which are mandatory:

```
- (void) didConnectSwiper:(WPYSwiper *)swiper;
- (void) didDisconnectSwiper:(WPYSwiper *)swiper;
- (void) didFailToConnectSwiper:(WPYSwiper *)swiper;
- (void) willConnectSwiper:(WPYSwiper *)swiper;
- (void) swiper:(WPYSwiper *)swiper didFailWithError:(NSString *)error;
- (void) swiper:(WPYSwiper *)swiper didFinishTransactionWithResponse:(WPYPaymentResponse *)response;
- (void) swiper:(WPYSwiper *)swiper didFailRequest:(WPYPaymentRequest *)request withError:(NSError *)error;
```

Step 3 – Get Card Information and Process Transaction

The IPC SDK interacts with the card reader to prompt the cardholder through confirming the amount of purchase and capturing payment card information, either through inserting the card in the chip reader, or by swiping it and reading the track data. Your application simply has to initiate the transaction, and the IPC SDK takes care of the rest:

```
WPYSwiper *swiper = [[WorldpayAPI instance] swiperWithDelegate:delegate;

[swiper connectSwiperWithInputType:WPYSwiperInputTypeBluetooth];

// After "didConnectToSwiper:" callback

WPYPaymentAuthorize *request = [[WPYPaymentAuthorize alloc] init];

request.amount = [NSDecimalNumber decimalNumberWithString:@"10.00"];

// Sale with cashback enabled. Use WPYEMVTransactionTypeGoods for sale without cashback enabled.
WPYEMVTransactionType type = WPYEMVTransactionTypeCashback;

// This handles both contact and magnetic stripe. If a chip enabled card is used, the SDK will prevent processing
// unless there are no supported Application IDs or if the chip fails to read and requires technical fallback.
[swiper beginEMVTransactionWithRequest:request transactionType:type];

// Handle payment errors:
- (void)swiper:(WPYSwiper *)swiper didFailRequest:(WPYPaymentRequest *)request withError:(NSError *)error
{
}

// Handle processed payment
- (void) swiper:(WPYSwiper *)swiper didFinishTransactionWithResponse:(WPYPaymentResponse *)response;
{
    // Clear the screen of the "Approved" or "Declined" message when ready
    [swiper swiperClearDisplay];
}
```

Step 4 – Handle the Response

When the transaction response is received from the Integrated Payment Hub, WPYSwiper will call either the `didFinishTransactionWithResponse()` or the `didFailRequest()` methods on the `WPYSwiperDelegate` you provided. You can look in the returned `WPYPaymentResponse` for information related to the transaction such as whether it was approved or declined. The `responseMessage` attribute provides a detailed description of the response.

Additional Reference Information

You can find more information from the following sources:

The developer website:
http://worldpay.us/developers

The Worldpay US Integrated Payments Client GitHub page:
http://github.com/worldpayus

The Worldpay IPC SDK for iOS Sample Application:
http://github.com/worldpayus/ipc_sdk_ios/SampleApp

