BE PAYMENT READY

Apple Pay In-App SDK and Apple Pay on the Web SDK - Merchant Integration Guide

Version: 1.0.2

Table of Contents

1 About This Documentation

This document contains instructions and specifications for using the Moneris Gateway Apple Pay In-App SDK and Apple Pay on the Web SDK to integrate your Apple Pay In-App and Apple Pay on the Web solutions with the Moneris Gateway.

1.1 System and Skills Requirements

For both Apple Pay In-App and Apple Pay on the Web:

- Register for an Apple Developer account on the Apple Developer Portal at https://developer-.apple.com/apple-pay/
- Refer to Apple's own documentation on their developer portal as well as this guide

For Apple Pay In-App:

- XCode 6.3 or higher
- Knowledge of Objective C or Swift
- iOS 8.0 or higher

For Apple Pay on the Web

- Safari browser on compatible Apple devices only
- Knowledge of Javascript

1.2 Getting Help

Moneris has help for you at every stage of the integration process.

Getting Started	During Development	Production
Contact our Client Integration Specialists: clientintegrations@moneris.com Hours: Monday – Friday, 8:30am to 8 pm ET	If you are already working with an integration specialist and need technical development assistance, contact our eProducts Technical Consultants: 1-866-319-7450 eproducts@moneris.com Hours: 8am to 8pm ET	If your application is already live and you need production support, contact Moneris Customer Service: onlinepayments@moneris.com 1-866-319-7450 Available 24/7

For additional support resources, you can also make use of our community forums at

http://community.moneris.com/product-forums/

2 Getting Started With Apple Pay

In order to integrate your Apple Pay In-App or Apple Pay on the Web payment solution, there are a few basic tasks you have to do to begin:

1. Developing a demo shopping cart application for Apple Pay in order to test functionality
2. Customizing your project's code to work with the Moneris Gateway
3. Boarding your Apple Pay credentials with Moneris

2.1 Developing an Apple Pay Demo App

In order to test the functionality of your Apple Pay solution with the Moneris Gateway, you first need a demo shopping cart application. Apple provides examples of demo applications for Apple Pay on the Apple Developer Portal for developers to use in integrating their Apple Pay solutions.

To build an Apple Pay In-App demo app example, follow the steps at:

https://developer.apple.com/library/content/samplecode/Emporium/Introduction/Intro.html

To build an Apple Pay on the Web demo app example, follow the steps at:

https://developer.apple.com/library/content/samplecode/EmporiumWeb/Introduction/Intro.html

2.2 Integrating Your Demo App With Moneris Gateway

In order for your Apple Pay payment application to use the Moneris Gateway, you need to customize your application code.

In the Apple Pay In-App demo app example, you modify the following files:

`ViewController.swift`

In the Apple Pay on the Web demo app example, you modify the following files:

`index.js`

`index.html`

2.2.1 Integrating Your Demo App – Apple Pay on the Web

To integrate your demo payment application for Apple Pay on the Web with the Moneris Gateway, do the following:

1. Go to Apple's developer portal to download the code for the EmporiumWeb app at: https://developer.apple.com/library/content/samplecode/EmporiumWeb/Introduction/Intro.html

2. In the **index.html** file, add the following code in the head section:

```
<script type="text/javascript" src="https://esqa.moneris.com/applepay/applepay-
   api.js"></script>
```

3. In the body of **index.html**, insert the following to enable the library to communicate:

```
<div id="moneris-apple-pay" store-id="store 1" merchant-
    identifier="merchant.com.moneris.apwebtest1" display-name="your-display-name"></div>
```

4. In **session.onpaymentauthorized**, insert the following code:

```
/**
* Payment Authorization
* Here you receive the encrypted payment data. You would then send it
* on to your payment provider for processing, and return an appropriate
* status in session.completePayment()
*/
session.onpaymentauthorized = (event) => {
// Send payment for processing...
const payment = event.payment;
const paymentJson = JSON.stringify(payment);
var request = {
orderId:"ApplePayWebTest"+((new Date()/1000)),
payment:event.payment
}
if (typeof(window.MonerisApplePay.purchase) === "function") {
window.MonerisApplePay.purchase(request, function(receipt) {
if (receipt.receipt.ResponseCode && !isNaN(receipt.receipt.ResponseCode)
&& parseInt (receipt.receipt.ResponseCode) < 50) {
session.completePayment(ApplePaySession.STATUS_SUCCESS);
}
else {
session.completePayment(ApplePaySession.STATUS_FAILURE);
}
});
}
}
```

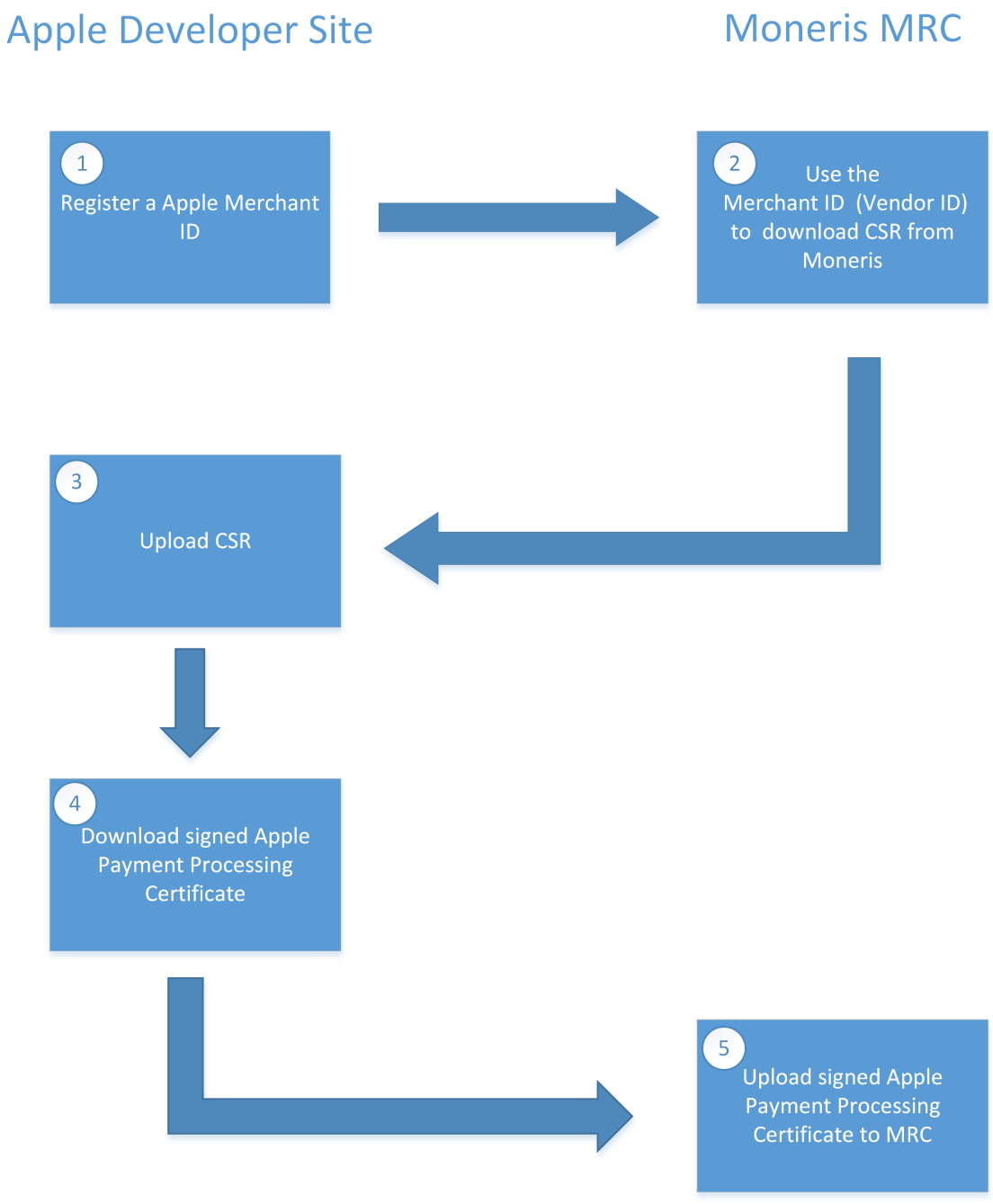
2.3 Boarding Your Apple Pay Solution

To ensure that your Apple Pay solution integrates securely with the Moneris Gateway, you need to obtain and upload signed credentials with both Apple and Moneris. You use the Moneris Merchant Resource Center in the boarding process.

The process for boarding differs slightly between the Apple Pay In-App and Apple Pay on the Web solutions.

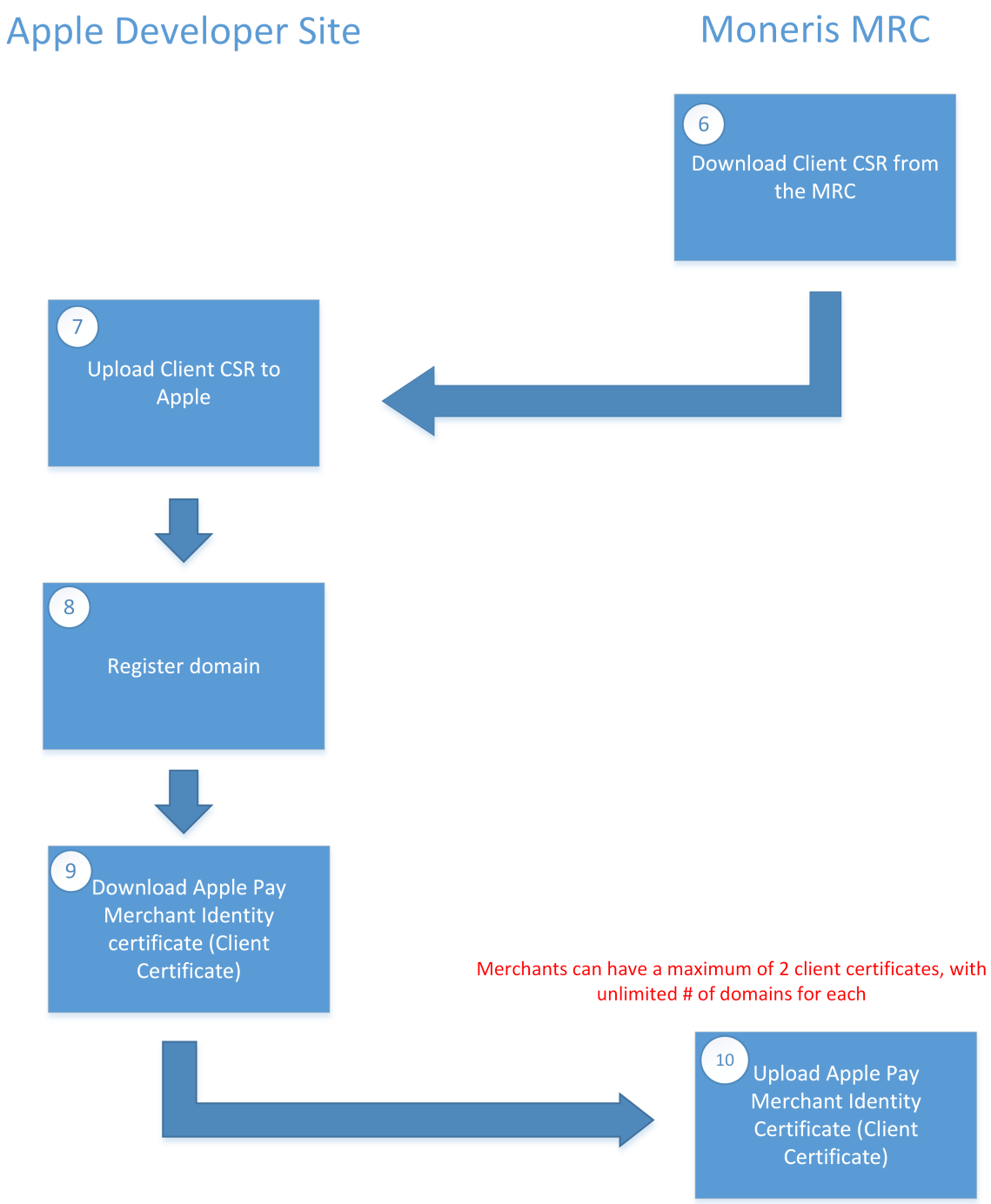
2.3.1 Boarding Process Flow for Apple Pay

Boarding Process - Applies to Apple Pay In-App and Apple Pay on the Web



2.3.2 Boarding Process Flow for Apple Pay on the Web

Additional Boarding Steps - Applies to Apple Pay on the Web ONLY



2.3.3 Boarding Your Apple Pay Solution – First Steps

To board your credentials for an Apple Pay solution, there are five basic initial steps for both Apple Pay In-App and Apple Pay on the Web:

1. Registering an Apple merchant ID at the Apple Developer Portal
2. Downloading a CSR file from the Moneris Merchant Resource Center
3. Uploading the CSR file to Apple
4. Downloading a signed Apple Pay Payment Processing Certificate from Apple
5. Uploading the signed Apple Pay Payment Processing Certificate to the Merchant Resource Center

The above steps complete the boarding process for Apple Pay In-App.

For boarding an Apple Pay on the Web solution, there are additional steps to perform.

2.3.3.1 Registering an Apple Merchant ID

The first required step for the boarding process for your Apple Pay In-App and Apple Pay on the Web is to get a Apple merchant ID on the Apple Developer Portal at developer.apple.com.

Once you have registered an Apple merchant ID, the next step is to go to the Moneris Merchant Resource Center to get a client signing certificate (CSR).

2.3.3.2 Downloading a CSR From Merchant Resource Center

Use your Apple merchant ID, also referred to as your Vendor ID, to obtain a client signing certificate (CSR) from the Moneris Merchant Resource Center.

To download your CSR from the Merchant Resource Center:

1. Go the Moneris Merchant Resource Center at one of the following URLs, depending on your stage of development:
   Testing/QA: https://esqa.moneris.com/mpg

   Production: https://www3.moneris.com/mpg
2. On the navigation bar at the top, select Admin > Apple Pay
3. In the fields under the Download CSR section, enter your Apple merchant ID and an email address as a point of contact
4. Click **Download** CSR to get the certification signing request (CSR) file from Moneris

The next step is to go to the Apple Developer Portal and upload the CSR you just downloaded.

2.3.3.3 Uploading the CSR to Apple

Once you have downloaded the CSR from the Moneris Merchant Resource Center, you upload it to the Apple Developer Portal at developer.apple.com.

The CSR from Moneris is required before Apple will issue you a Apple Pay Payment Processing Certificate in the next step.

NOTE: This procedure or some of its details may change at the discretion of Apple, please refer to the Apple Developer Portal for the most up-to-date information.

To upload the CSR to the Apple Developer Portal:

1. In the Apple Developer Portal, go to Identifiers > Merchant IDs
2. Under the Apple Pay Payment Processing Certificate section, click Create Certificate
3. In the Generate your certificate step, choose the CSR file from its location and click Upload

Once the CSR is uploaded, the next step is to download the signed Apple Payment Processing Certificate.

2.3.3.4 Downloading a Signed Apple Pay Payment Processing Certificate

After you have uploaded your Moneris CSR to the Apple Developer Portal, Apple gives you the option to download the signed Apple Pay Payment Processing Certificate.

NOTE: This procedure or some of its details may change at the discretion of Apple, please refer to the Apple Developer Portal for the most up-to-date information.

To download the signed Apple Pay Payment Processing Certificate, click **Download** and save it to your device.

The next step is to upload this certificate to the Moneris Merchant Resource Center.

2.3.3.5 Uploading the Signed Certificate to MRC

Once you have the signed Apple Pay Payment Processing Certificate, you upload it to the Merchant Resource Center in order to complete the boarding process for the Apple Pay In-App solution.

If you are boarding an Apple Pay on the Web solution, you must do additional steps for boarding.

To upload the signed Apple Pay Payment Processing Certificate to the Merchant Resource Center:

1. Select **Admin > Apple Pay** in the Merchant Resource Center
2. Under the heading Apple Merchant Certificates, find the row with your Vendor ID (i.e., the Apple merchant ID)
3. Click the **Upload Apple Signed Certificate** button in that row
4. Choose the certificate from its location on your device to upload it.

2.3.4 Additional Steps for Boarding Apple Pay on the Web

In addition to following the first five steps described in Boarding Your Apple Pay Solution – First Steps, boarding your Apple Pay on the Web solution requires further steps as follows:

6. Downloading a Client CSR file from the Moneris Merchant Resource Center
7. Uploading the Client CSR file to the Apple Developer Portal
8. Registering a payment processing domain with Apple
9. Downloading an Apple Pay Merchant Identity Certificate
10. Uploading the Apple Pay Merchant Identity Certificate (Client Certificate) to the Merchant Resource Center

2.3.4.1 Downloading a Client CSR from MRC

The first additional step in the Apple Pay on the Web boarding process is to download a Client CSR from the Moneris Merchant Resource Center.

To download a Client CSR:

1. In the Merchant Resource Center, go to Admin > Apple Pay
2. Under the section Apple Merchant Certificates, find the row that contains your Vendor ID (Apple merchant ID)
3. Click Client Certificates
4. In the email field, enter your email address as a point of contact
5. In the Domain field, add the domain that you will register for processing with Apple
6. Click Download Client CSR and save the Client CSR file to your device

Once you have downloaded a Client CSR, the next step is to upload the Client CSR at Apple's Developer Portal.

2.3.4.2 Uploading a Client CSR to Apple

To obtain an Apple Pay Merchant Identity Certificate for your Apple Pay on the Web solution, you upload the Moneris Client CSR that you downloaded from the Merchant Resource Center.

2.3.4.3 Registering Your Domain with Apple

The Apple Pay on the Web solution requires you to register with Apple each web domain where you will be performing Apple Pay transactions.

> **NOTE:** This procedure or some of its details may change at the discretion of Apple, please refer to the Apple Developer Portal for the most up-to-date information.

To register your domain with Apple:

1. Go to the Apple Developer Portal and select your merchant ID
2. Click **Edit**
3. Under Merchant Domains in the Apple Pay Payment Processing on the Web section, click **Add Domain**

Once you have registered a domain, the next step is to download the Apple Pay Merchant Identity Certificate.

2.3.4.4 Downloading Apple Pay Merchant Identity Certificate

The Apple Pay Merchant Identity Certificate is referred to in the Moneris Merchant Resource Center as the Client Certificate.

To download the Apple Pay Merchant Identity Certificate:

1. Go to the Apple Developer Portal and select your merchant ID
2. Click **Edit**
3. Under Apple Pay Merchant Identity Certificate , click **Download**

Once you have downloaded the Apple Pay Merchant Identity Certificate, the final step is to upload it to the Moneris Merchant Resource Center.

2.3.4.5 Uploading Apple Pay Merchant Identity Certificate

Once you have downloaded your Apple Pay Merchant Identity Certificate, the final step is to upload that certificate to the Moneris Merchant Resource Center, where it is referred to as a "Client Certificate".

To upload the Apple Merchant Identity Certificate (Client Certificate) to the MRC:

1. In the Merchant Resource Center, go to Admin > Apple Pay
2. Under the Apple Merchant Certificates section, find the row containing your merchant ID (Vendor ID)
3. Click **Client Certificate**
4. In the Apple Client Certificate section of the new page, find the row listed for your domain
5. Click **Upload Apple Signed Certificate** to upload the Apple Merchant Identity Certificate file on your device to the MRC

3 Transaction Types

Moneris Gateway Apple Pay SDK supports the following transactions:

- Apple Pay Token Purchase (AppleTokenPurchase)
- Apple Pay Token Pre-Authorization (AppleTokenPreauth)

> **NOTE:** INTERAC® e-Commerce transaction functionality is currently available only when processing a Purchase transaction.

In addition, the Moneris Gateway Apple Pay SDK also supports optional features, such as Customer Information and recurring transactions.

Once you have processed the initial transaction using AppleTokenPurchase or AppleTokenPreauth, you can use the Moneris Gateway standard API set available on the Developer Portal at

https://developer.moneris.com

to process one of the following supplemental transactions:

- Refund
- Purchase Correction
- Pre-Authorization Completion

3.1 Apple Pay Token Purchase

The Apple Pay Token Purchase transaction verifies funds on the customer's card, removes the funds and readies them for deposit into the merchant's account.

Required transaction objects – Apple Pay Token Purchase

Object	Apple Pay on the Web (JavaScript)
Order ID	`orderId:document.getElementById('order-id').value`
ApplePayPaymentToken	`payment:event.payment`

Optional objects

Table 1: Optional Transaction Feature Objects - Apple Pay

Function Type	Description	Class Name | Parameters
Customer Information	Customer details such as shipping, taxes etc can be sent with a financial trans-actions such as AppleTokenPurchase and AppleTokenPreauth	-(void)setCustInfoWithPayment NSString *shippingAddress, NSString *instructions , NSString *shippingCost, NSString *tax1, NSString *tax2, NSString *tax3
Recurring billing	Recurring Billing is feature which allows for a trans-action information to be sent once and then rebilled on a specified interval for a certain number of times.	-(void)setRecur NSString *recurUnit NSString *startNow NSString *startDate NSString *numRecurs NSString *period NSString *recurAmount

3.2 Apple Pay Token Pre-Authorization

The Apple Pay Token Pre-Authorization verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time, based on the card issuer. A subsequent Completion trans-action must be performed for the funds to settle into the merchant's account.

This transaction can only be performed on a credit card.

Required transaction objects – Apple Pay Token Pre-Authorization

Object	Apple Pay on the Web (JavaScript)
Order ID	`orderId:document.getElementById('order-id').value`
ApplePayPaymentToken	`payment:event.payment`

Optional transaction objects

Table 1: Optional Transaction Feature Objects - Apple Pay

Function Type	Description	Class Name | Parameters
Customer Information	Customer details such as shipping, taxes etc can be sent with a financial transactions such as AppleTokenPurchase and AppleTokenPreauth	-(void)setCustInfoWithPayment NSString *shippingAddress, NSString *instructions , NSString *shippingCost, NSString *tax1, NSString *tax2, NSString *tax3
Recurring billing	Recurring Billing is feature which allows for a transaction information to be sent once and then rebilled on a specified interval for a certain number of times.	-(void)setRecur NSString *recurUnit NSString *startNow NSString *startDate NSString *numRecurs NSString *period NSString *recurAmount

3.3 Performing Follow-On Transactions

After you have performed an Apple Pay transaction, you can subsequently perform other 'follow-on' transactions:

- Refund
- Purchase Correction, also known as a void
- Pre-Authorization Completion

For more about these transactions, refer to the Moneris Developer Portal at:

https://developer.moneris.com/Documentation/NA/E-Commerce%20Solutions/API

4 Testing Your Solution – Apple Pay

Testing your Apple Pay integration differs according to whether you are implementing the Apple Pay In-App or Apple Pay on the Web solution.

Apple Pay In-App – see 4.3 Testing Your Solution – Apple Pay In-App
Apple Pay on the Web – see 4.4 Testing Your Solution – Apple Pay on the Web

4.1 Getting a Unique Test Store ID and API Token

Transactions requests via the Moneris Gateway API will require you to have a Store ID and a corresponding API token.

> **NOTE:** The API token method is not used for the Apple Pay on the Web solution.

For testing purposes, you can either use the pre-existing test stores, or you can create your own unique test store where you will only see your own transactions.

To get your unique Store ID and API token for testing:

1. Log in to the Developer Portal at https://developer.moneris.com
2. In the My Profile dialog, click the **Full Profile** button
3. Under My Testing Credentials, select **Request Testing Credentials**
4. Enter your Developer Portal password and select your country
5. Record the Store ID and API token that are given, as you will need them for logging in to the Merchant Resource Center (Store ID) and for API requests (API token).

Alternatively, you can use the pre-existing test stores already set up in the Merchant Resource Center as described in 4.2 Test Store Credentials.

For production, you will use the Store ID given to you in your Moneris activation letter and an API token retrieved from the production Merchant Resource Center. For more on this, see 5.1 Getting a Production Store ID and API Token.

4.2 Test Store Credentials

For testing purposes, you can either use the pre-existing test stores, or you can create your own unique test store where you will only see your own transactions. If you want to use the pre-existing stores, use the test credentials provided in the following tables.

Table 1: Test Server Credentials - Canada

Store ID	API Token	MRC Username	MRC Password
store1	yesguy	demouser	password
store2	yesguy	demouser	password
store3	yesguy	demouser	password
store4	yesguy	demouser	password
store5	yesguy	demouser	password

Alternatively, you can create and use a unique test store where you will only see your own transactions. For more on this, see 4.1 Getting a Unique Test Store ID and API Token

4.3 Testing Your Solution – Apple Pay In-App

For testing your Apple Pay In-App solution, you will need to:

- **Complete the boarding process** as described in 2.3.3 Boarding Your Apple Pay Solution – First Steps
- **Get test Store ID and test API token** – get these on the Moneris Developer Portal at https://developer.moneris.com; see 4.1 Getting a Unique Test Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see 2.3.3.1 Registering an Apple Merchant ID
- **Configure code in the MpgRequest object** – see 4.3.1 Configuring MpgRequest Object for Testing

4.3.1 Configuring MpgRequest Object for Testing

To configure the **MpgRequest** object in your Apple Pay In-App solution for testing:

1. Insert your production Store ID and production API token
2. Change the `setTestMode` value to `NO`

Comparison of the MpgRequest object code between testing and production is shown below.

Table 1: MpgRequest Object Comparison - Testing vs. Production

Testing/Development	Production
`MpgRequest *req = [MpgRequest mpgRequestWithStoreId:@"moneris" ApiToken:@"hurgle" Transaction:transaction]; [req setTestMode:YES]; [req setProcCountry:@"CA"];`	`MpgRequest *req = [MpgRequest mpgRequestWithStoreId:@"PRODUCTION STOREID PROVIDED BY MONERIS" ApiToken:@"PRODUCTION API TOKEN PROVIDED BY MONERIS" Transaction:transaction]; [req setTestMode:NO]; [req setProcCountry:@"CA"];`

4.4 Testing Your Solution – Apple Pay on the Web

For testing your Apple Pay on the Web solution you need to:

- **Complete the boarding process** as described in section 2.3 Boarding Your Apple Pay Solution
- **Get test Store ID** – get this on the Moneris Developer Portal at https://developer.moneris.com; see 4.1 Getting a Unique Test Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see 2.3.3.1 Registering an Apple Merchant ID
- **Configure code in the index.html file** – see 4.4.1 Configuring index.html File for Testing

4.4.1 Configuring index.html File for Testing

To configure the **index.html** file in your Apple Pay on the Web solution for testing:

1. In the **index.html** file, change the `script` tag's `src` attribute to link to the test library location:
```
<script

type="text/javascript"

src="https://esqa.moneris.com/applepay/applepay-api.js">

</script>
```

2. In the body of **index.html**, change the `div` tag's `store-id` and `merchant-identifier` attributes to reflect your test Store ID and Apple merchant ID:
```
<div

 id="moneris-apple-pay"

 store-id="store 1"

 merchant-identifier="merchant.com.moneris.apwebtest1">

</div>
```

5 Moving to Production – Apple Pay

Putting your Apple Pay solution into production differs according to whether you are implementing the Apple Pay In-App or Apple Pay on the Web solution.

Apple Pay In-App – see 5.2 Configuring for Production – Apple Pay In-App
Apple Pay on the Web – see 5.3 Configuring for Production – Apple Pay on the Web

5.1 Getting a Production Store ID and API Token

In production, you use the Store ID that was given in your activation letter from Moneris. You obtain the production API token from the production Merchant Resource Center.

> **NOTE:** The API token method is not used in the Apple Pay on the Web solution.

To get your production API token:

1. If you have not already done so, activate your production account at

   https://www.moneris.com/activate

   The activation process provides you with your first administrator user for the Merchant Resource Center.

2. Once activated, log in to the production Merchant Resource Center at

   https://www3.moneris.com/mpg

3. Select the **Admin** menu and choose **Settings**. Your production API token is located under the API token heading on the page.

5.2 Configuring for Production – Apple Pay In-App

To move your Apple Pay on the Web solution into production you need to:

- **Complete the boarding process** as described in 2.3 Boarding Your Apple Pay Solution
- **Get production Store ID** – see 5.1 Getting a Production Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see 2.3.3.1 Registering an Apple Merchant ID
- **Configure code in the MpgRequest object** – see 5.3.1 Configuring index.html File for Production

5.2.1 Configuring MpgRequest Object for Production

To configure the **MpgRequest** object in your Apple Pay In-App solution for production:

1. Insert your test Store ID and test API token
2. Change the `setTestMode` value to `YES`

Comparison of the MpgRequest object code between testing and production is shown below.

Table 1: MpgRequest Object Comparison - Testing vs. Production

Testing/Development	Production
`MpgRequest *req = [MpgRequest mpgRequestWithStoreId:@"moneris" ApiToken:@"hurgle" Transaction:transaction]; [req setTestMode:YES]; [req setProcCountry:@"CA"];`	`MpgRequest *req = [MpgRequest mpgRequestWithStoreId:@"PRODUCTION STOREID PROVIDED BY MONERIS" ApiToken:@"PRODUCTION API TOKEN PROVIDED BY MONERIS" Transaction:transaction]; [req setTestMode:NO]; [req setProcCountry:@"CA"];`

5.3 Configuring for Production – Apple Pay on the Web

To move your Apple Pay on the Web solution into production you need to:

- **Complete the boarding process** as described in 2.3 Boarding Your Apple Pay Solution
- **Get production Store ID** – see 5.1 Getting a Production Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see 2.3.3.1 Registering an Apple Merchant ID
- **Configure code in the index.html file** – see 5.3.1 Configuring index.html File for Production

5.3.1 Configuring index.html File for Production

To configure the **index.html** file in your Apple Pay on the Web solution for production:

1. In the **index.html** file, change the `script` tag's `src` attribute to link to the production library location:
```
<script

type="text/javascript"

src="https://www3.moneris.com/applepay/applepay-api.js">

</script>
```
2. In the body of **index.html**, change the `div` tag's `store-id` and `merchant-identifier` attributes to reflect your production Store ID and production Apple merchant ID:
```
<div

 id="moneris-apple-pay"
```

```
 store-id="store 1"
 merchant-identifier="merchant.com.moneris.apwebtest1">
</div>
```

6 Verifying Your Transactions

To verify your transactions have processed:

1. Log in to the Merchant Resource Center at https://esqa.moneris.com/mpg (testing) or https://www3.moneris.com/mpg (production)
2. Find your transactions under **Reports > Transactions**

Appendix A Mpg Transaction Request Properties

Variable	Size/Type	Description
Store ID `storeId`	Apple Pay In-App: NSString, 10-character alpha-numeric Apple Pay on the Web: String, 10-character alpha-numeric	This is defined and provided by Moneris Solutions and is used to identify the merchant
API Token `apiToken`	Apple Pay In-App: NSString, 20-character alpha-numeric	This is defined and provided by Moneris Solutions and is used in conjunction with the Store ID to uniquely identify your store. **NOTE:** API Token is not used with Apple Pay In-Browser.
Host `host`	alphanumeric	Development host URL: https://esqa.moneris.com Production host URL: https://www3.moneris.com

Appendix B Definition of Request Object Fields

Table 1: Definition of Request Fields - Apple Pay

Variable Name	Size/Type	Description
orderId	10-character alphanumeric	Merchant defined unique transaction identifier - must be unique for every Purchase, Authorization and Independent Refund attempts, regardless if they approved or declined. Must also be unique per merchant account. For a follow-on transaction (Refunds, Completions, and Purchase Corrections) the order_id must reference the original transaction.
Amount	9-character decimal (variable length)	Amount of the transaction This must contain at least 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 999999.99
payment	PassKit Payment	Apple Pay payment object returned in the didAuthorizePayment method This is passed in the Apple Pay Purchase or Pre-authorization request
avs_street_number avs_street_name	19-character alphanumeric	Street Number & Street Name (max. 19-digit limit for street number and street name combined) Must match the address the issuing bank has on file

Variable Name	Size/Type	Description
avs_zipcode	10-character alphanumeric	Zip or Postal Code Must match the address the issuing bank has on file
cvd_value	4-character numeric	Credit Card CVD value – this number accommodates either 3 or 4 digit CVD values **NOTE:** The CVD value supplied by the cardholder should simply be passed to the Moneris Gateway. Under no circumstances should it be stored for subsequent uses or displayed as part of the receipt information.
cvd_indicator	1-character numeric	CVD presence indicator Typically the value is 1 Possible values: 0 - CVD value is deliberately bypassed or is not provided by the merchant. 1 - CVD value is present. 2 - CVD value is on the card, but is illegible. 9 - Cardholder states that the card has no CVD imprint

B.1 Customer Information Fields for Apple Pay

Table 1: Customer Information Request Fields - Apple Pay

Field Name	Size/Type
first name	30-character alphanumeric
last name	30-character alphanumeric
company name	50-character alphanumeric

Field Name	Size/Type
address	70-character alphanumeric
city	30-character alphanumeric
province	30-character alphanumeric
postal code	30-character alphanumeric
country	30-character alphanumeric
phone	30-character alphanumeric
fax	30-character alphanumeric
tax1	10-character alphanumeric
tax2	10-character alphanumeric
tax3	10-character alphanumeric
shipping_cost	10-character alphanumeric
Extra Details	
email	60-character alphanumeric
instructions	100-character alphanumeric

Appendix C Definition of Response Fields

Table 1: Definition of Response Variables - Apple Pay

Variable	Size/Type	Description
ReceiptId	50-character alphanumeric	The order id specified in the request will be echoed back in the response. This field is recommended to be displayed on the receipt for tracking and troubleshooting purposes.
ReferenceNum	18-character numeric	This is a bank transaction reference number. The entire reference number must be displayed on the receipt. This information should also be stored by the merchant. The following illustrates the breakdown of this field where "660123450010690030" is the reference number returned in the message. 660123450010690030 • 66012345: Terminal ID • 001: Shift number • 069: Batch number • 003: Transaction number within the batch.
ReponseCode	3-character numeric	Transaction Response Code < 50: Transaction approved >= 50: Transaction declined NULL: Transaction was not sent for authorization Custom Apple Pay Response Code 900 : Global Error means that Moneris Gatewaywas unable to decrypt payload.

Variable	Size/Type	Description
ISO	2-character numeric	ISO response code
AuthCode	8-character alphanumeric	Authorization code returned from the issuing institution
TransTime	HH:II:SS	Processing host time stamp. Time of the transaction. Must be displayed on the transaction receipt. HH = 2-digit hour, 24 hour clock (“0” left padded | 02 = 2am, 14 = 2pm) II = 2-digit minute (“0” left padded) SS = 2-digit seconds (“0” left padded
TransDate	YYYY-MM-DD	Processing host date stamp. Date of the transaction. Must be displayed on the transaction receipt. YYYY = 4-digit year MM = 2-digit month (“0” left padded | Jan = 01) DD = 2 digit day of month (“0” left padded)
TransType	alphanumeric	Type of transaction that was performed 00 = Purchase 01 = Pre-authorization
Complete	true/false	Transaction was sent to authorization host and a response was received
Message	100-character alphanumeric	Response description returned from issuing institution
TransAmount	nnnnnnN.NN	Returns the amount sent in

Variable	Size/Type	Description
	9-character decimal (variable length)	request for processing. The amount represents the amount that the cardholder was charged/refunded. The amount must be displayed on the receipt. charged/refunded. The amount must be displayed on the receipt. **NOTE:** The amount will always contain one (1) dollar value and two (2) cent values separated by a period ".". N = always returned n = returned when required
TxnNumber	20-character alphanumeric	Gateway Transaction identifier
CardType	2-character alphanumeric	Credit Card Type M = MasterCard V = Visa AX = American Express P = INTERAC®
TimedOut	true/false	Transaction failed due to a process timing out
Bank Totals	Object	Response data returned in a Batch Close and Open Totals request.
Ticket	n/a	Reserved field
CorporateCard	true/false	Indicates whether the card is a corporate card or not
CAVV	40-character alphanumeric	Decrypted CAVV value for the transaction Returned for Apple Pay Purchase/Pre-Authorization transaction if payload is successfully decrypted

Variable	Size/Type	Description
IsVisaDebit	true/false/null	Indicates whether the card that the transaction was performed on is Visa debit true = Card is Visa Debit false = Card is not Visa Debit null = there was an error in identifying the card
RecurSuccess	true/false	Indicates whether the transaction successfully registered
DeviceManufacturerIdentifier	12-character alphanumeric	Token requestor ID. Returned from decrypted payload. Hex-encoded device manufacturer identifier.
AvsResultCode	1-character alphanumeric	Indicates the address verification result
CvdResultCode	2-character alphanumeric	Indicates the CVD validation result

Appendix D Recurring Billing

Recurring Billing allows you to set up payments whereby Moneris automatically processes the transactions and bills customers on your behalf based on the billing cycle information you provide.

> **Things to Consider:**
>
> - To avoid shifting, do not set the `start_date` after the 28th if the `recur_unit` is `month`. To set the billing date for the last day of the month, set `recur_unit` to `eom`.

D.1 Setting Up a New Recurring Payment

In addition to instantiating a transaction object and a HttpsPostRequest connection object, you must instantiate a Recur object. This object has a number of mandatory properties that must be set.

Recur Object Definition

Table 2: Recur object mandatory arguments

Value	Type	Limits	Variable Name
	Description		
Recur unit	String	`day`, `week`, `month` or `eom`	`recur_unit`
	Unit to be used as a basis for the interval. This can be set as day, week, month or the end of the month. Works in conjunction with the `period` argument (see below) to define the billing frequency.		
Start Now	String	`true`/`false`	`start_now`
	If a single charge is to be made against the card immediately, set this value to `true`. The amount to be billed immediately may differ from the amount billed on a regular basis thereafter. If the billing is to start in the future, set this value to `false`.		

Table 2: Recur object mandatory arguments

Value	Type	Limits	Variable Name
	Description		
Start Date	String	YYYY/MM/DD format	`start_date`
	Date of the first future recurring billing transaction. This value **must** be a date in the future. If an additional charge is to be made immediately, the `start_now` argument must be set to `true`.		
Number of Recurs	String	numeric 1-99	`num_recurs`
	The number of times that the transaction must recur.		
Period	String	numeric 1-999	`period`
	Number of recur units that must pass between recurring billings.		
Recurring Amount	String	9-character decimal 0.01-9999999.99.	`recur_amount`
	Amount of the recurring transaction. This must contain at least three digits, two of which are penny values. This is the amount that will be billed on the `start_date`, and then billed repeatedly based on the interval defined by `period` and `recur_unit`.		

Given a Recur object with the above syntax, D.1 shows how the transaction is interpreted for different argument values.

Table 3: Recurring Billing examples

Argument	Values	Description
recur_unit	"month";	The first transaction occurs on January 2, 2030 (because start_now="false"). The card is billed $30.00 every 2 months on the 2nd of each month. The card will be billed a total of 12 times. This includes the transaction on January 2, 2030
start_date	"2030/01/02"	
num_recurs	"12"	
start_now	"false"	
period	"2"	
recur_amount	"30.00"	
recur_unit	"week";	The first charge is billed immediately (because start_now-w=true). The initial charge is $15.00. Beginning on January 2, 2030 the credit card will be billed $30.00 every 2 weeks for 26 recurring charges. Therefore, the card will be billed a total of 27 times. (1 immediate and 26 recurring.)
start_date	"2030/01/02"	
num_recurs	"26"	
start_now	"true"	
period	"2"	
recur_amount	"30.00"	

Appendix E Error Messages

Error messages that are returned if the gateway is unreachable

Global Error Receipt

You are not connecting to our servers. This can be caused by a firewall or your internet connection.

Response Code = NULL

The response code can be returned as null for a variety of reasons. The majority of the time, the explanation is contained within the Message field.

When a 'NULL' response is returned, it can indicate that the issuer, the credit card host, or the gateway is unavailable. This may be because they are offline or because you are unable to connect to the internet.

A 'NULL' can also be returned when a transaction message is improperly formatted.

Error messages that are returned in the Message field of the response

XML Parse Error in Request: <System specific detail>
An improper XML document was sent from the API to the servlet.

XML Parse Error in Response: <System specific detail>
An improper XML document was sent back from the servlet.

Transaction Not Completed Timed Out
Transaction timed out before the host responds to the gateway.

Request was not allowed at this time
The host is disconnected.

Could not establish connection with the gateway: <System specific detail>
Gateway is not accepting transactions or server does not have proper access to internet.

Input/Output Error: <System specific detail>
Servlet is not running.

The transaction was not sent to the host because of a duplicate order id
Tried to use an order id which was already in use.

The transaction was not sent to the host because of a duplicate order id
Expiry Date was sent in the wrong format.

Vault error messages

Can not find previous
Data key provided was not found in our records or profile is no longer active.

Invalid Transaction
Transaction cannot be performed because improper data was sent.

or

Mandatory field is missing or an invalid SEC code was sent.

Malformed XML
Parse error.

Incomplete

Timed out.

or

Cannot find expiring cards.

Appendix F Security Requirements

All Merchants and Service Providers that store, process, or transmit cardholder data must comply with PCI DSS and the Card Association Compliance Programs. However, validation requirements vary by business and are contingent upon your "Merchant Level" or "Service Provider Level". Failure to comply with PCI DSS and the Card Association Compliance Programs 3.2 may result in a Merchant being subject to fines, fees or assessments and/or termination of processing services. Non-compliant solutions may prevent merchants boarding with Moneris Solutions.

As a Moneris Solutions client or partner using this method of integration, your solution must demonstrate compliance to the Payment Card Industry Data Security Standard (PCI DSS) and/or the Payment Application Data Security Standard (PA DSS) 3.2. These standards are designed to help the cardholders and merchants in such ways as they ensure credit card numbers are encrypted when transmitted/stored in a database and that merchants have strong access control measures, logging, secure software updates, secure remote access and support.

For further information on PCI DSS and PA DSS requirements, please visit www.pcisecuritystandards.org.

For more information on how to get your application PCI-DSS compliant, please contact our Integration Specialists and visit https://developer.moneris.com to view the PCI-DSS Implementation Guide.