BE PAYMENT READY

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

Version: 1.1.2

Table of Contents

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
    - Apple Pay In-App: https://developer.apple.com/in-app-purchase/
    - Apple Pay on the Web: https://developer.apple.com/documentation/apple_pay_on_the_web

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

Changes in v1.1.2

- Added CustID to the code sample in Integrating Your Website – Apple Pay on the Web
- Added optional head and style code to "Integrating Your Website – Apple Pay on the Web" on page 13. This code enables Apple Pay support for IOS 18 on third-party browsers such as Google Chrome and Microsoft Edge.

Changes in 1.1.1 August 2nd, 2023

- Updated setup flow for In Web to match the Advanced menu options within the Merchant Resource Center
- Updated Integrating Your Website - Apple Pay on the Web with new code samples
- Added applePayRequest line to code sample, for use in scenarios where a merchant wishes to have their line items echo-back in the Moneris response, as ApplePay does not include line item details in their own response

Changes in 1.1.0

- Added Preload transaction type information for Apple Pay on the Web
- Added additional information to indicate the amount request field represents transaction amount after shipping is calculated
- Added change log to document

1 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. Boarding your Apple Pay credentials with Moneris
2. Developing a demo shopping cart application for Apple Pay In-App or a website with payment page for Apple Pay on the Web in order to test functionality
3. Customizing your project's code to work with the Moneris Gateway

1.1 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 Simple Setup and Advanced Setup options:

- **Simple Setup** is straightforward and does not require signed certificates, but only supports Apple Pay on the Web solutions
- **Advanced Setup** supports both Apple Pay In-Appand Apple Pay on the Web

1.1.1 Simple Setup

This setup method for Apple Pay is straightforward and easy, but only allows in-browser payments. If you need in-app payments, you will need to generate and upload Apple Pay certificates. See "Advanced Setup" on the next page for the full process.

> **NOTE:** For testing a Simple Setup Apple Pay configuration, you will need a sandbox account. See "Getting a Unique Test Store ID and API Token" on page 22

> **Things to Consider:**Non-profit or charitable organizations must register with Apple's partner Benevity before they may use the Simple configuration steps below.

1. Go the MonerisMerchant 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. Click on the **Add Profile** button

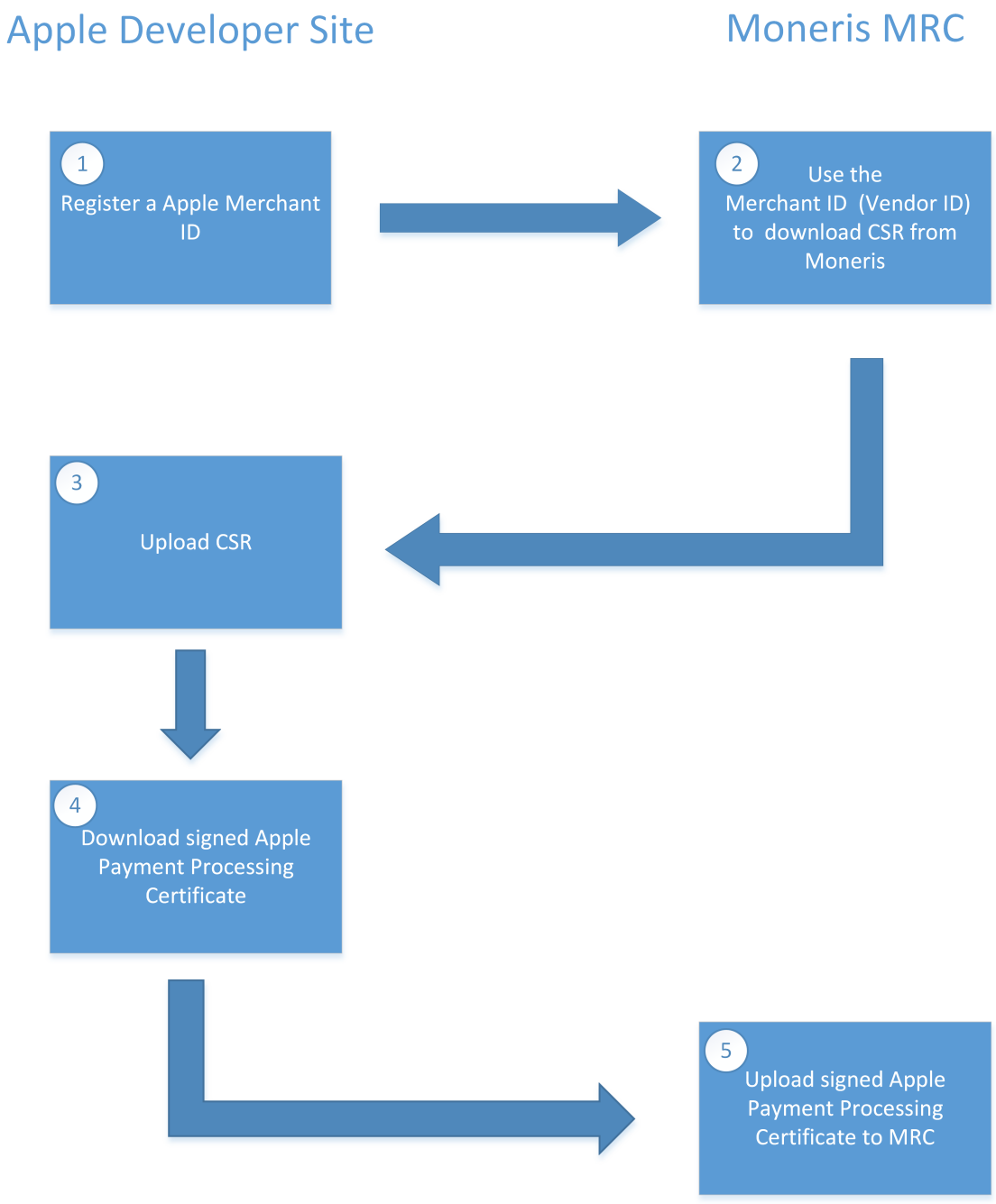
   If this button is missing, your account is flagged as a non-profit or charitable organization.Please register with Benevity or use the Advanced Setup instead.
4. Enter your website's domain name for registration and click on the **Next** button
5. Click on the **Download** button to obtain a file Apple uses to verify your ownership of the domain. You will need to place this file at a specified location.
6. Apple will expect the file at the https://yourdomain.ca/.well-known/apple-developer-merchantid-domain-association location. Move the file there before proceeding.
7. Return to the Merchant Resource Centerand click on the **Verify** button.

1.1.2 Advanced Setup

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.

1.1.2.1 Boarding Process Flow for Apple Pay

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



1.1.2.2 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.

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).

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. Click on the **Advanced Setup** button
4. Click on the **Add Profile** button and enter your Apple Merchant Identifier
5. Locate your Apple Merchant Identifier in the list and click on the **Edit** button
6. Click **Download Payment Processing 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.

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 Pay payment processing certificate.

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.

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. Click on the **Advanced Setup** button
3. Under the heading Apple Merchant Identifiers, find the row with your Vendor ID (i.e., the Apple Merchant Identifier aka Apple merchant ID) and click on **Edit**
4. Click the **Upload Payment Processing Certificate** button
5. Choose the certificate from its location on your device to upload it.

1.1.2.3 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

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. Click on the **Advanced** button
3. Under the section Apple Merchant Identifiers, find the row that contains your Vendor ID (Apple Merchant Identifier, aka Apple Merchant ID)
4. Click **Edit**
5. Click **Download Merchant Identity 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.

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.

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.

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.

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 "Merchant Identity 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. Click on the **Advanced** button
3. Under the Apple Pay Merchant Identity Certificates section, find the row containing your merchant ID (Vendor ID)
4. Click **Edit**
5. In the Apple Client Certificate section of the new page, click **Upload Merchant Identity Certificate** to upload the Apple Merchant Identity Certificate file on your device to the MRC

1.2 Developing an Apple Pay Demo App or Payment Page

In order to test the functionality of your Apple Pay solution with the Moneris Gateway, you first need a demo shopping cart application or payment page.

Apple provides some examples for Apple Pay In-App and Apple Pay on the Web on the Apple developer portal for developers to use in integrating their Apple Pay solutions.

For an example of a Apple Pay In-App demo app example:

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

For a payment page example for Apple Pay on the Web:

https://applepaydemo.apple.com/

1.3 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`

1.4 Integrating Your Website – Apple Pay on the Web

To integrate your website's payment page with Apple Pay on the Web via the Moneris Gateway, do the following:

1. Configure your back-end server to send a Preload request to Moneris Gateway in order to get a unique ticket number for each transaction; for more info on Preload requests, see 2.1 Preload Request
2. (Optional) On your payment page, add the following code in the head and style sections to enable Apple Pay support for IOS 18 on Third Party Browsers (i.e, alternatives to Safari, such as Google Chrome or Microsoft Edge)
   Head

   ```
   <head> <script crossorigin src="https://applepay.cdn-apple.com/jsapi/1.latest/apple-pay-sdk.js"> </script> </head>
   ```

   Style

   ```
   <style> apple-pay-button { --apple-pay-button-width: 140px; --apple-pay-button-height: 30px; --apple-pay-button-border-radius: 5px; --apple-pay-button-padding: 5px 0px; } </style> <apple-pay-button buttonstyle="black" type="buy" locale="el-GR"></apple-pay-button>
   ```

3. On your payment page, add the following code in the head section:
   Testing/QA

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

   Production

   ```
   <script type="text/javascript" src="https://www3.moneris.com/applepayjs/applepay-api.js"></script>
   ```

4. In the body of your payment page, insert the following `<div>` to enable the library to communicate:
```
<div id="moneris-apple-pay" store-id="store 1" merchant-identifier="your-unique-merchant-identifier" display-name="your-display-name"></div>
```

5. In the Apple Pay event handler **session.onpaymentauthorized**, configure the code so it looks like the following:
```
const request = {
                        countryCode: 'CA',
                        currencyCode: 'CAD',
                        supportedNetworks: ['visa', 'masterCard', 'amex', 'discover'],
                        merchantCapabilities: ['supports3DS'],
                        total: { label: 'Moneris Production Test', type: "final", amount:
'35.00' },
                        "requiredBillingContactFields": [
                                "postalAddress",
                                "name"
                        ],
                        "requiredShippingContactFields": [
                                "postalAddress",
                                "name",
                                "phone",
                              "email"
                              ],
                        "lineItems": [
                              {
                                    "label": "Donation",
                                    "amount": "20.00"
                              },
                              {
                                    "label": "Membership",
                                    "amount": "15.00"
                              }
                        ]
}

session = new ApplePaySession(3, request);

window.MonerisApplePay.setTicket(ticket);
window.MonerisApplePay.setCustID("nqa-cust");


session.onvalidatemerchant = (event) => {
      console.log("Validate merchant");
      const validationURL = event.validationURL;
    window.MonerisApplePay.validateSession(validationURL, apSessionValidationSuccess,
apSessionValidationError);
};

session.onpaymentauthorized = (event) => {
      // Send payment for processing...
      const payment = event.payment;
      const paymentJson = JSON.stringify(payment);
      const respDiv = document.getElementById("apple-pay-resp");
      const uiDiv = document.getElementById("apple-pay-ui");

      uiDiv.classList.add("none");

      respDiv.innerText = "Processing Payment...Please wait";

      var moneris_request = {
            payment:event.payment,
            applePayRequest: request// optional - to be added to receive cust info
      }

      var transType = "preauth";
      if (typeof(window.MonerisApplePay[transType]) === "function") {
```

```
                window.MonerisApplePay[transType](moneris_request, function(receipt) {
                      //make call to your backend to check async response
                      console.log(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);
                      }
                      respDiv.innerText = "" + JSON.stringify(receipt);
                });
          }
    }

    session.begin();
    }
    function apSessionValidationSuccess(response) {
          console.log(response);
          session.completeMerchantValidation(response);
    }
    function apSessionValidationError(response) {
          console.log("Failed to validate merchant session");
          console.log(response);
```

NOTE: Merchants who wish to receive an echo back of customer information, including line items, can also include:

```
applePayRequest: request
```

If you will be processing INTERAC® payments via Apple Pay on the Web, there is an additional step. See 1.5 Enabling INTERAC® e-Commerce Transactions.

1.5 Enabling INTERAC® e-Commerce Transactions

If your merchant application will be processing INTERAC® e-Commerce debit transactions through Apple Pay, you will need to do additional steps to ensure this feature works.

Before processing INTERAC® e-Commerce transactions, you will also need to have that functionality enabled by your Moneris sales representative.

To enable your Apple Pay integration to process INTERAC® e-Commerce transactions:

1. Ensure your merchant account with Moneris has INTERAC® e-Commerce functionality turned on; your Moneris sales representative can do this for you
2. In your payment page code, under the function **applePayButtonClicked()**, insert `'interac'` into the code as shown:
   `supportedNetworks:[ 'amex', 'discover', 'masterCard', 'visa', 'interac'],`

For more information on this subject, see Apple's documentation for Apple Pay JS API on the Apple developer portal at https://developer.apple.com/documentation/apple_pay_on_the_web/apple_pay_js_api.

2 Transaction Types

Moneris Gateway Apple Pay SDK supports the following transactions:

- Apple Pay Purchase
- Apple Pay Pre-Authorization

Once you have processed the initial transaction using Apple Pay Purchase or Apple Pay Pre-Authorization, you can use the Moneris Gateway Unified API or Batch Upload solution to process one of the following supplemental transactions:

- Refund
- Purchase Correction (not applicable to INTERAC® e-Commerce transactions)
- Pre-Authorization Completion

Information about the Unified API and Batch Upload are available on the Moneris Developer Portal at

developer.moneris.com

> **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.

2.1 Preload Request

Requests a unique ticket number from the Moneris Gateway server, which responds with the ticket number to your back-end server at the URL provided.

The Preload request is sent prior to sending a financial transaction in Apple Pay on the Web and Google Pay™ Web.

If no response is received within 15 seconds, the request will time out.

Preload transaction request URL for Moneris Gateway

Testing: https://esqa.moneris.com/gateway2/servlet/MpgRequest

Production: https://www3.moneris.com/gateway2/servlet/MpgRequest

Preload transaction request object

<applepay_preload>

Connection fields – Required

Variable Name	Type and Limits	Description
store ID <store_id>	*String* N/A	Unique identifier provided by Moneris upon merchant account setup
API token <api_token>	*String* N/A	Unique alphanumeric string assigned by Moneris upon merchant account activation To find your API token, refer to your test or production store's Admin settings in the Merchant Resource Center, at the following URLs: Testing: https://esqa.moneris.com/mpg/ Production: https://www3.moneris.com/mpg/

Preload transaction request fields – Required

Variable Name	Type and Limits	Description
order ID <order_id>	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	Merchant-defined transaction identifier that must be unique for every Purchase, Pre-Authorization and Independent Refund transaction. No two transactions of these types may have the same order ID. For Refund, Completion and Purchase Correction transactions, the order ID must be the same as that of the original transaction.
amount <amount>	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point	Transaction dollar amount This must contain at least 3 digits, two of which are penny values Minimum allowable value = $0.01, maximum allowable value = $9999999.99

Variable Name	Type and Limits	Description
	EXAMPLE: 1234567.89	**NOTE:** This amount is the total transaction amount after shipping is calculated
receipt URL <receipt_url>	*String*	URL that the Moneris Gateway will send your receipt to

2.1.1 Example Preload Transaction Request

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
    <store_id>your_storeId</store_id>
    <api_token>your_apiToken</api_token>
    <applepay_preload>
        <order_id>example_orderId</order_id>
        <amount>1.00</amount>
        <receipt_url>https://example.com/example.php</receipt_url>
    </applepay_preload>
</request>
```

2.1.2 Example Preload Transaction Response

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <receipt>
        <PreloadTicket>ap1575405989TfehNuPgCwi3BBNEXN</PreloadTicket>
        <Message>Preload request successfully registered</Message>
        <ResponseCode>001</ResponseCode>
    </receipt>
</response>
```

2.2 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.

Apple Pay Token Purchase transaction request fields – Required

Variable Name	Apple Pay on the WebJavaScript	Description
ticket	`ticket='obtain_ticket_using_Preload'`	Unique transaction identifier issued by the Moneris Gateway Ticket number will always start with **ap**
payment	`payment:event.payment`	Encrypted payload information

Optional objects (Apple Pay In-App only)

Function Type	Class Name | Parameters	Description
Customer Information	-(void)setCustInfoWithPayment NSString *shippingAddress, NSString *instructions , NSString *shippingCost, NSString *tax1, NSString *tax2, NSString *tax3	Customer details such as shipping, taxes, etc. can be sent with a financial transactions such as Apple Pay Token Purchase and Apple Pay Token Pre-Authorization
Recurring billing	-(void)setRecur NSString *recurUnit NSString *startNow NSString *startDate NSString *numRecurs NSString *period NSString *recurAmount	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.

2.3 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 Pre-Authorization Completion transaction must be performed for the funds to settle into the merchant's account.

This transaction can only be performed on a credit card.

Apple Pay Token Pre-Authorization transaction request fields – Required

Variable Name	Apple Pay on the WebJavaScript	Description
ticket	`ticket='obtain_ticket_using_Preload'`	Unique transaction identifier issued by the Moneris Gateway Ticket number will always start with **ap**
payment	`payment:event.payment`	Encrypted payload information

Optional transaction objects (Apple Pay In-App only)

Function Type	Class Name | Parameters	Description
Customer Information	-(void)setCustInfoWithPayment NSString *shippingAddress, NSString *instructions , NSString *shippingCost, NSString *tax1, NSString *tax2, NSString *tax3	Customer details such as shipping, taxes, etc. can be sent with a financial transactions such as Apple Pay Token Purchase and Apple Pay Token Pre-Authorization
Recurring billing	-(void)setRecur NSString *recurUnit NSString *startNow NSString *startDate NSString *numRecurs NSString *period NSString *recurAmount	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.

2.4 Performing Follow-On Transactions

After you have performed a 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

3 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 3.3 Testing Your Solution – Apple Pay In-App
Apple Pay on the Web – see 3.4 Testing Your Solution – Apple Pay on the Web

3.1 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 with the corresponding test API tokens, or you can create your own unique test API token and a 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 3.2 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 4.1 Getting a Production Store ID and API Token.

3.2 Test Store Credentials

For testing purposes, you can either use the pre-existing test stores with the corresponding test API tokens, or you can create your own unique test API token and a unique test store where you will only see your own transactions. If you want to use 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 3.1 Getting a Unique Test Store ID and API Token

3.3 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 1.1.2.2 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 3.1 Getting a Unique Test Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see Registering an Apple Merchant ID
- **Configure code in the MpgRequest object** – see 3.3.1 Configuring MpgRequest Object for Testing

3.3.1 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"];`

3.4 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 1.1 Boarding Your Apple Pay Solution
- **Get test Store ID** – get this on the Moneris Developer Portal at https://developer.moneris.com; see 3.1 Getting a Unique Test Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see Registering an Apple Merchant ID
- **Configure code on your payment page** – see 3.4.1 Configuring Your Payment Page for Testing

> **NOTE:** For each transaction that is processed, you should verify the payment amount to ensure that all data passed to Moneris in the request was correct. For more information, see 5 Verifying Your Transactions.

3.4.1 Configuring Your Payment Page for Testing

To configure your payment page in your Apple Pay on the Web solution for testing:

1. On your payment page, change the `script` tag's `src` attribute to link to the test library location:
```
<script

type="text/javascript"

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

</script>
```

2. In the body of your payment page, 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="your-unique-QA-merchant-identifier">

</div>
```

4 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 4.2 Configuring for Production – Apple Pay In-App
Apple Pay on the Web – see 4.3 Configuring for Production – Apple Pay on the Web

4.1 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.

4.2 Configuring for Production – Apple Pay In-App

To move your Apple Pay In-App solution into production you need to:

- **Complete the boarding process** as described in 1.1 Boarding Your Apple Pay Solution
- **Get production Store ID** – see 4.1 Getting a Production Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see Registering an Apple Merchant ID
- **Configure code in the MpgRequest object** – see 4.3.1 Configuring Your Payment Page for Production

4.2.1 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"];`

4.3 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 1.1 Boarding Your Apple Pay Solution
- **Get production Store ID** – see 1 Getting a Production Store ID and API Token
- **Get Apple merchant ID** – get this at the Apple Developer Portal; see Registering an Apple Merchant ID
- **Configure code in the index.html file** – see 4.3.1 Configuring Your Payment Page for Production

> **NOTE:** For each transaction that is processed, you should verify the payment amount to ensure that all data passed to Moneris in the request was correct. For more information, see 5 Verifying Your Transactions.

4.3.1 Configuring Your Payment Page for Production

To configure your payment page in your Apple Pay on the Web solution for production:

1. In your payment page code, change the `script` tag's `src` attribute to link to the production library location:

```
<script

type="text/javascript"

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

```
</script>
```

2. In the body of your payment page, 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="your-unique-production-merchant-identifier">

</div>
```

5 Verifying Your Transactions

When you send a Preload transaction, you must include a transaction amount and a receipt URL, hosted on your back-end server. The Moneris Gateway sends the transaction response to your receipt URL so that you can verify the amount you sent in the transaction request matches the one coming back in the response from the Moneris Gateway.

As a secondary check, you can also manually look at your transactions in the Moneris Merchant Resource Center.

To use the Merchant Resource Center 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 Name	Type and Limits	Description
store ID `storeId`	Apple Pay In-App: NSString, 10-character alphanumeric Apple Pay on the Web: String, 10-character alpha-numeric	Unique identifier provided by Moneris upon merchant account setup
API token `apiToken`	Apple Pay In-App: NSString, 20-character alphanumeric	Unique alphanumeric string assigned by Moneris upon merchant account activation To find your API token, refer to your test or production store's Admin settings in the Merchant Resource Center, at the following URLs: Testing: https://esqa.moneris.com/mpg/ Production: https://www3.moneris.com/mpg/ **NOTE:** API token is not used with Apple Pay In-Browser.
host `host`	*String* alphanumeric	Development host URL: https://esqa.moneris.com Production host URL: https://www3.moneris.com

6 Setting Up a New Recurring Payment

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`.

Recurring Billing Info 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`.		
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.		

Table 2: Recur object mandatory arguments

Value	Type	Limits	Variable Name
	Description		
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, 6 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"`	

Argument	Values	Description
recur_unit	"week";	The first charge is billed immediately (because start_now=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 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
avs_zipcode	10-character alphanumeric	Zip or Postal Code Must match the address the issu-

Variable Name	Size/Type	Description
		ing 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
address	70-character alphanumeric
city	30-character alphanumeric

Field Name	Size/Type
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

Variable Name	Type and Limits	Description
AuthCode	*String* 8-character alphanumeric	Authorization code returned from the issuing institution
Bank Totals	*Object* N/A	Response data returned in a Batch Close and Open Totals request.
CardType	*String* 2-character alphanumeric	Credit Card Type M = Mastercard V = Visa AX = American Express P = INTERAC®
CAVV	*String* 40-character alphanumeric	Decrypted CAVV value for the transaction Returned for Apple Pay Purchase/Pre-Authorization transaction if payload is successfully decrypted
Complete	*String* true/false	Transaction was sent to authorization host and a response was received
CorporateCard	*String* true/false	Indicates whether the card is a corporate card or not
ISO	*String* 2-character numeric	ISO response code
IsVisaDebit	*String* 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

Variable Name	Type and Limits	Description
Message	*String* 100-character alpha-numeric	Response description returned from issuing institution
ReceiptId	*String* 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.
RecurSuccess	*String* true/false	Indicates whether the transaction successfully registered
ReferenceNum	*String* 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. **EXAMPLE** 660123450010690030 • 66012345: Terminal ID • 001: Shift number • 069: Batch number • 003: Transaction number within the batch
ReponseCode	*String* 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 Name	Type and Limits	Description
Ticket	N/A	Reserved field
TimedOut	*String* true/false	Transaction failed due to a process timing out
TransAmount	*String* nnnnnnN.NN 9-character decimal (variable length)	Returns the amount sent in request for processing. The amount represents the amount that the cardholder was 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
TransDate	*String* 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)
TransID	*String* 20-character alphanumeric	Gateway Transaction identifier
TransTime	*String* 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)
TransType	*String* numeric	Type of transaction that was performed 00 = Purchase 01 = Pre-authorization

C.1 Apple Pay Specific Response Fields

Variable Name	Type and Limits	Description
AvsResultCode	*String* 1-character alphanumeric	Indicates the address verification result
CvdResultCode	*String* 2-character alphanumeric	Indicates the CVD validation result
DeviceManufacturerIdentifier	*String* 12-character alpha-numeric	Token requestor ID. Returned from decrypted payload. Hex-encoded device manufacturer identifier.

Appendix D Preload Response Codes

The following codes are returned by the Moneris Gateway in response to Preload transactions

Response Code	Message
001	Preload request successfully registered
900	Global Error
901	Invalid URL
902	Malformed XML

7 Response Codes

Approved Response Codes

Response Code	Messages
000	Approved, Account Balances Included (Balance Inquiry), No Reason to Decline Approved (Balances) File Processed/Successful transaction with fault
001	Approved, Account Balances Not Included Approved – No Balances/Approved or completed successfully VIP Approved (No Balances)/Advice Acknowledged – Financial Liability Accepted
002	Approved, country club
003	Approved, maybe more ID
004	Approved, pending ID (sign paper draft)
005	Approved, blind
006	Approved, VIP
007	Approved, administrative transaction
008	Approved, national NEG file hit OK
009	Approved, commercial
010	Approved for partial amount
023	Amex - credit approval
024	Amex 77 - credit approval
027	Transaction already reversed

Response Code	Messages
028	VIP Credit Approved
029	Credit Response Acknowledgement
900	Global Error
901	Invalid URL
902	Malformed XML

Declined Response Codes

Response Code	Messages
050	Do Not Honor Decline Refer to card issuer ID certification fails Deny – Do not Honour Card not initialized Declined: Deny – Unacceptable Fee Unable to locate original transaction Suspected Fraud Deny – Card Acceptor Call Acquirer’s Security Dep Amount Not Reconciled – Totals Provided ATM/POS terminal number cannot be located MAC failed Declined: MAC failed Reserved Security processing failure No arrears (transaction receipt not printed)

Response Code	Messages
	Invalid File Type No such File File Locked Unsuccessful Incorrect File Length File Decompression Error File Name Error File cannot be received Deny – Do Not Honour
051	Expired Card
052	PIN retries exceeded PIN try limit exceeded Allowable number of PIN tries exceeded
053	No sharing
054	No security module
055	Invalid transaction
056	No Support/Transaction Not Permitted to Acquirer Tran Not Supported by FI/Not Supported by Receiver
057	Lost or stolen card
058	Invalid status
059	Deny (Keep Card) – Restricted Card Restricted Card
060	No Chequing account No Savings Account
061	No PBF

Response Code	Messages
062	PBF update error
063	Invalid authorization type
064	Bad Track 2
065	Adjustment not allowed
066	Invalid credit card advance increment
067	Invalid transaction date
068	PTLF error
069	Bad Message Error/No CVM Results Bad message – edit error/Format error
070	No IDF Invalid Issuer Invalid Issuer/Deny – Issuer/Bank Not Found
071	Invalid route authorization Unable to route/Financial institution or intermediate network facility cannot be found for routing Invalid Rout to Auth /Incorrect IIN
072	Card on National NEG file
073	Invalid route service (destination)
074	Unable to authorize Re-enter Transaction Transaction Cannot be Completed Deny – Security Violation Deny – Violation of Law System problem - ask cardholder to insert card in chip card reader Merchant Link not logged on (Network Management Logon required)
075	Invalid PAN length

Response Code	Messages
076	Low funds
077	Pre-auth full
078	Duplicate transaction Duplicate transaction/Request in progress
079	Maximum online refund reached
080	Maximum offline refund reached
081	Maximum credit per refund reached
082	Number of times used exceeded
083	Maximum refund credit reached
084	Duplicate transaction - authorization number has already been corrected by host
085	Inquiry not allowed
086	Over floor limit
087	Maximum number of refund credit by retailer
088	Place call
089	CAF status inactive or closed
090	Referral file full
091	NEG file problem
092	Advance less than minimum
093	Delinquent
094	Over table limit
095	Amount over maximum Amt Over Max/Transaction amount limit exceeded
096	PIN required

Response Code	Messages
097	Mod 10 check failure
098	Force Post
099	Bad PBF

Referral Response Codes

Response Code	Messages
100	Unable to process transaction Invalid Request. Contact Moneris Client POS Certification for repeat declines. Network Unavailable System Malfunction
101	Place call
102	Refer – Call Expired Card Card Acceptor Contact Call Card Accpt Acq Secur
103	NEG file problem
104	CAF problem
105	Card not supported
106	Amount over maximum
107	Over daily limit
108	CAF Problem
109	Advance less than minimum
110	Number of times used exceeded
111	Delinquent

Response Code	Messages
112	Over table limit
113	Timeout
115	PTLF error
121	Administration file problem
122	Unable to validate PIN: security module down

System Error Response Codes

Response Code	Messages
150	Invalid Service Code/Merchant Merchant Not On File Merchant Not on File/Invalid Merchant
200	Invalid account Invalid Card Number Invalid Account/Deny – No Account Type Requested
201	Incorrect PIN Invalid PIN/Incorrect personal identification number PIN Block Error
202	Advance less than minimum
203	Administrative card needed
204	Amount over maximum
205	Invalid Advance Amount Original Amnt Incorrect Bad message/Invalid Amount Original transaction amount error
206	CAF not found

Response Code	Messages
	Invalid “to” account Invalid “from” account Invalid account
207	Invalid transaction date
208	Invalid expiration date
209	Invalid transaction code
210	PIN key sync error
212	Destination not available
251	Error on cash amount
252	Debit not supported

American Express Response Codes (Declines)

Response Code	Messages
426	AMEX - Denial 12
427	AMEX - Invalid merchant
429	AMEX - Account error
430	AMEX - Expired card
431	AMEX - Call Amex
434	AMEX - Call 03 Note: Invalid CVD (CID)
435	AMEX - System down
436	AMEX - Call 05
437	AMEX - Declined
438	AMEX - Declined

Response Code	Messages
439	AMEX - Service error
440	AMEX - Call Amex
441	AMEX - Amount error

Credit Card Response Codes (Declines)

Response Code	Messages
408	CREDIT CARD - Card use limited - Refer to branch
475	CREDIT CARD - Invalid expiration date
476	CREDIT CARD - Invalid transaction, rejected No Credit Account Invalid transaction/Invalid related transactions Unable to process/Suspected malfunction; related transaction error Unable to Authorize: Cut off is in process Issuer not capable to process Switch system malfunction Issuer response not received by CUPS Unable to Authorize/Illegal Status of Acquirer
477	CREDIT CARD - Refer Call/Invalid Card Number Invalid card number (no such account) Deny – Card Not Found Items not on Bankbook beyond limit, declined/Invalid card number
478	CREDIT CARD - Decline, Pick up card, Call
479	CREDIT CARD - Decline, Pick up card
480	CREDIT CARD - Decline, Pick up card
481	CREDIT CARD - Decline

Response Code	Messages
	Transaction not allowed to be processed by cardholder Low funds/Insufficient Balance Invalid Transaction Transaction not allowed to be processed by merchant
482	CREDIT CARD - Expired Card
483	CREDIT CARD – Refer/Refer to Issuer Deny – Card Acceptor Contact Acquirer
484	CREDIT CARD - Expired card - refer
485	CREDIT CARD - Not authorized
486	CREDIT CARD - CVV Cryptographic error
487	CREDIT CARD - Invalid CVV
489	CREDIT CARD - Invalid CVV
490	CREDIT CARD - Invalid CVV
492	System problem - ask cardholder to insert card in chip card reader Withdrawal count exceeded

System Decline Response Codes

Response Code	Messages
800	Bad format
801	Bad data
802	Invalid Clerk ID
809	Bad close
810	System timeout
811	System error
821	Bad response length

Response Code	Messages
877	Invalid PIN block
878	PIN length error
880	Final packet of a multi-packet transaction
881	Intermediate packet of a multi-packet transaction
889	MAC key sync error
898	Bad MAC value
899	Bad sequence number - resend transaction
900	Capture - PIN Tries Exceeded
901	Capture - Expired Card
902	Capture - NEG Capture
903	Capture - CAF Status 3
904	Capture - Advance < Minimum
905	Capture - Num Times Used
906	Capture - Delinquent
907	Capture - Over Limit Table
908	Capture - Amount Over Maximum Capture - Capture Pick up Card Suspected Fraud Hard Capture Deny – Keep Card: Special Conditions Expired Card Fraud Card Acceptor Call Acquirer’s

Response Code	Messages
	Do Not Honour
950	Admin card is not enabled on Merchant profile

Other Response Codes

Response Code	Message
599	Decline

Admin Response Codes

Response Code	Messages
960	Initialization Failure - No Match on Merchant ID
961	Initialization Failure - No Match on PED ID
962	Initialization Failure - No match on Printer ID
963	No match on Poll code
964	Initialization Failure - No match on Concentrator ID
965	Invalid software version number
966	Duplicate terminal name
970	Terminal/Clerk table full
983	Clerk Totals Unavailable: selected Clerk IDs do not exist or have zero totals
989	MAC Error on Transaction 95 (Initialization and Handshake), most often, this indicates that the wrong keys have been injected into a device/KMAC Sync Error

EMV Reversal Request Codes

Response Code	Messages
990	Chip card declines a host approved transaction

Response Code	Messages
991	Chip card removed before ICC communications are completed

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.