Merchant Integration Guide - Canada
Apple Pay SDK - iOS

V1.0.0

Revision Number	**Date**	**Description**
V1.0.0	July 7, 2016	Initial Draft

Table of Contents

1. Introduction

This documentation contains the instructions for integrating ApplePay as an in-app purchase option using Canada iOS SDK. It contains specifications for transaction requests and responses. Also contained herein is information on obtaining the requirements for Credit card transaction receipts.

2. System and Skill Requirements

1. XCode 6.3 or higher
2. Knowledge of Objective C
3. iOS 8.0 or higher

3. Essential Integration Tasks

- In order to create a Merchant ID and obtain Merchant certificate please refer to the link below for instructions https://developer.moneris.com
- Configurations and settings of Xcode in order for the code to compile: Enable Apple Pay in Xcode by selecting designated project -> Capabilities pane -> Enable Apple Pay & in- App purchase -> Select the Merchant Id
- To configure the payment sheet with additional information please refer to https://developer.apple.com/apple-pay/. Apple Pay displays a payment sheet with the contact information, shipping, and payment related information to the customer before the customer authorizes the payment for processing.
- The ‘buy with apple pay’ button can also be configured using the PassKit Framework Reference. Please refer to https://developer.apple.com/apple-pay/ for details.

4. Function Types

Moneris Gateway Apple Pay SDK supports the following types of transactions ApplePayTokenPreauth , ApplePayTokenPurchase & CAVV Purchase through the MpgClasses . Below are the list of all the supported transactions in conjunction with other extra features such as Cust info, AVS, CVD and Recurring info.

Function Type	Description	Class Name | Parameters
Financial Transactions		
AppleTokenPurchase	The Purchase transaction verifies funds on the customer's card, removes the funds and readies them for deposit into the merchant's account.	**@interface ApplePayTokenPurchase** NSString *orderId, NSString *payment
AppleTokenPreauth	The 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 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.*	**@interface ApplePayTokenPreauth** NSString *orderId, NSString *payment
CavvPurchase	CAVVPurchase transaction is performed using CAVV value obtained by performing a VBV/MasterCard SecureCode transaction on card. Verifies the validity of the CAVV value, Removes the funds and readies them for deposit into merchant account.	@interface CavvPurchase NSString *orderId, NSString *amount, NSString *pan, NSString *expDate, NSString *cryptType, NSString *cavv
Optional Methods		
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 Information	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,

CVD Information	Card Validation digits information can be passed with the Financial Transaction to be verified for a particular transaction.	-(void)setCvdInfo NSString *cvdIndicator, NSString *cvdValue
AVS Information	Address Verification information can be passed with the Financial Transaction to be verified for a particular transaction	-(void)setAvsInfo NSString *streetNumber, NSString *streetName, NSString *zipCode

5. Code Sample

Sample

```
//
//  ViewController.m
//  projnorth
//
//  Created by Spencer Lai on 2015-04-22.
//  Copyright (c) 2015 Spencer Lai. All rights reserved.
//


#import "ViewController.h"
@import PassKit;
@import AddressBook;
#import <MpgClasses/MpgClasses.h>
#import <MpgClasses/ApplePayTokenPurchase.h>
#import <MpgClasses/ApplePayTokenPreauth.h>

@interface ViewController ()

@end

@implementation ViewController
@synthesize recurUnit;
@synthesize hasRecur;
@synthesize recurAmount;
@synthesize recurStartDate;
@synthesize recurStartNow;
@synthesize txtAmount;
@synthesize btnTransType;
@synthesize configured;
@synthesize transAmount;

NSArray * transTypes;

- (void)viewDidLoad
{
    [super viewDidLoad];

    if (!self.configured)
    {
        self.selectedIdx = -1;
        self.configured = YES;
        [self.txtAmount addTarget:self action:@selector(txtAmountDidChange:)
forControlEvents:UIControlEventValueChanged];
    }

    transTypes = [NSArray arrayWithObjects: @"Purchase", @"Preauth", nil];
    [self loadPayButton];
    self.txtReceipt.editable = NO;

    [self updateGui];

    self.paymentNetworks = [[NSArray alloc] initWithObjects:PKPaymentNetworkAmex,
PKPaymentNetworkVisa, PKPaymentNetworkMasterCard, nil];

    // Do any additional setup after loading the view, typically from a nib.
}
-(void)viewWillDisappear:(BOOL)animated
{
    self.transAmount = self.txtAmount.text;
}

-(void)txtAmountDidChange:(id)sender
{
    self.transAmount = self.txtAmount.text;
}
- (void)updateGui
{
    [self.txtAmount setText:self.transAmount];

    if (self.selectedIdx < [transTypes count] && self.selectedIdx >= 0)
    {
        [self.btnTransType setTitle:[transTypes objectAtIndex:self.selectedIdx]
forState:UIControlStateNormal];
    }
    else
    {
        [self.btnTransType setTitle:@"Select Transaction Type" forState:UIControlStateNormal];
    }

    [self updateRecurButton];
}

-(void)updateRecurButton
{
    if (self.hasRecur)
    {
        [self.btnRecur setTitle:@"Recur Enabled" forState:UIControlStateNormal];
        [self.btnRecur setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
    }
    else
    {
        [self.btnRecur setTitle:@"Recur Disabled" forState:UIControlStateNormal];
        [self.btnRecur setTitleColor:[UIColor grayColor] forState:UIControlStateNormal];
    }
}
- (void)loadPayButton
{
    CGFloat btnX = 15.0;
    CGFloat btnY = [UIScreen mainScreen].bounds.size.height - 60.0;
    CGFloat btnWidth = [UIScreen mainScreen].bounds.size.width - 30.0;
    CGFloat btnHeight = 50.0;

    self.btnPay = [PKPaymentButton buttonWithType:PKPaymentButtonTypeBuy
style:PKPaymentButtonStyleWhite];
    self.btnPay.frame = CGRectMake(btnX, btnY, btnWidth, btnHeight);
    [self.btnPay addTarget:self action:@selector(pommeButtonDidPress:)
forControlEvents:UIControlEventTouchUpInside];

    [self.view addSubview:self.btnPay];

}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (void) pommeButtonDidPress:(id)sender
{

    if ([PKPaymentAuthorizationViewController canMakePayments])
    {
        NSLog(@"Can pay using networks");
        if ([PKPaymentAuthorizationViewController canMakePayments])
        {
            NSLog(@"Can make payment... let's start...");
            self.request = [[PKPaymentRequest alloc] init];

            self.request.currencyCode = @"USD";
            self.request.countryCode = @"US";
//          self.request.merchantIdentifier = @"merchant.moneris.teststore";
            self.request.merchantIdentifier = @"merchant.moneris.caqateststore1";
            self.request.requiredShippingAddressFields = PKAddressFieldAll;

            self.request.merchantCapabilities = PKMerchantCapability3DS;
            self.request.supportedNetworks = @[PKPaymentNetworkAmex,PKPaymentNetworkMasterCard,
PKPaymentNetworkVisa];

            self.request.paymentSummaryItems = @[
                                                 [PKPaymentSummaryItem summaryItemWithLabel:@"Item"
amount:[NSDecimalNumber decimalNumberWithString:self.txtAmount.text]]
                                                 ];
            PKPaymentAuthorizationViewController *payView = [[PKPaymentAuthorizationViewController
alloc] initWithPaymentRequest:self.request];
            if (payView)
            {
                payView.delegate = self;

                [self presentViewController:payView animated:YES completion:nil];
            }
            else
            {

            }
        }
        else
        {
            NSLog(@"NONONO network!!!");
        }
    }
    else
    {
        NSLog(@"NONONO payment!!!");
    }
}

-(void)recurButtonDidPress:(id)sender
{
    self.transAmount = self.txtAmount.text;
    [self performSegueWithIdentifier:@"recurModal" sender:self];
}

-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"recurModal"])
    {
        if (self.recurInfoView == nil)
        {
            self.recurInfoView = [segue destinationViewController];

            [((UIRecurViewController *)self.recurInfoView) setRecurEnabled:self.hasRecur
startNow:[self.recurStartNow isEqualToString:@"true"] transAmount:self.transAmount
recurAmount:self.recurAmount startDate:self.recurStartDate numOfRecur:self.numOfRecurs
unit:self.recurUnit interval:self.recurInterval];
        }

    }
}

-(void)paymentAuthorizationViewController:(PKPaymentAuthorizationViewController *)controller
didAuthorizePayment:(PKPayment *)payment completion:(void
(^)(PKPaymentAuthorizationStatus))completion// instance method
{
    NSLog(@"payment authroized!!!");
    MpgTransaction *transaction;
    Recur *recurInfo;
    if (self.hasRecur)
    {
        NSLog(@"This transaction has recur!!!");

        recurInfo = [Recur recurWithUnit:self.recurUnit StartNow:self.recurStartNow
StartDate:self.recurStartDate NumberOfRecurs:[NSString stringWithFormat:@"%ld",self.numOfRecurs]
Period:[NSString stringWithFormat:@"%ld", self.recurInterval] RecurAmount:self.recurAmount];
    }

    ABMultiValueRef phoneRef = ABRecordCopyValue(payment.shippingAddress, kABPersonPhoneProperty);

    NSLog(@"There's %ld phone in phoneRef", ABMultiValueGetCount(phoneRef) );

    switch (self.selectedIdx) {
        case 0:
            transaction = (MpgTransaction *)[ApplePayTokenPurchase applePayTokenPurchaseWithOrderId:
[NSString stringWithFormat:@"Spence-PROJ_NORTH-%ld", (long)[[NSDate date] timeIntervalSince1970]]
                                                                                        Payment:payment];
            [((ApplePayTokenPurchase *)transaction) setCustInfoWithPayment:payment Instructions:@""
ShippingCost:@"" Tax1:@"" Tax2:@"" Tax3:nil];
            [((ApplePayTokenPurchase *)transaction) setCvdInfo:[CvdInfo cvdInfoWithIndicator:@"1"
Value:@"189"]];
            if (recurInfo != nil)
            {
                [((ApplePayTokenPurchase *)transaction) setRecur:recurInfo];
            }
            break;
        case 1:
            transaction = (MpgTransaction *)[ApplePayTokenPreauth applePayTokenPreauthWithOrderId:
[NSString stringWithFormat:@"Spence-PROJ_NORTH-%ld", (long)[[NSDate date] timeIntervalSince1970]]

Payment:payment];
            [((ApplePayTokenPreauth *)transaction) setCustInfoWithPayment:payment Instructions:@""
ShippingCost:@"" Tax1:@"" Tax2:@"" Tax3:nil];
            [((ApplePayTokenPreauth *)transaction) setCvdInfo:[CvdInfo cvdInfoWithIndicator:@"1"
Value:@"189"]];

            if (recurInfo != nil)
            {
                [((ApplePayTokenPreauth *)transaction) setRecur:recurInfo];
            }
            break;
        default:
            completion(PKPaymentAuthorizationStatusFailure);
            break;
    }


    MpgRequest *req = [MpgRequest mpgRequestWithStoreId:@"moneris" ApiToken:@"hurgle"
Transaction:transaction];
    [req setTestMode:YES];
    [req setProcCountry:@"CA"];

    HttpsPostRequest *post = [[HttpsPostRequest alloc] initWithHost:@"esqa.moneris.com" Request:req];

    MpgResponse *resp = post.response;

    self.txtReceipt.text = [NSString stringWithFormat:@"Receipt
ID:\t%@\nMessage:\t%@\nCAVV:\t%@\nResponse code:\t%@\nDevice Manufacturer Identifer:\t%@\n",
resp.getReceiptId, resp.getMessage, resp.getCavv, resp.getResponseCode,
resp.getDeviceManufacturerIdentifier];
    NSLog(@"%@", resp.getReceiptId);

    NSScanner *respCodeScanner = [NSScanner scannerWithString:resp.getResponseCode];

    if (([resp.getResponseCode integerValue] <=50) && ([respCodeScanner scanInt:NULL] &&
[respCodeScanner isAtEnd]))
    {
        completion(PKPaymentAuthorizationStatusSuccess);
    }
    else
    {
        completion(PKPaymentAuthorizationStatusFailure);
    }


}

-(void)transTypeButtonDidPress:(id)sender
{
    [self.txtAmount resignFirstResponder];
    UIActionSheet *transTypeSheet = [[UIActionSheet alloc] initWithTitle:@"Select Transactions Type"
                                                                delegate:self
cancelButtonTitle:@"Cancel" destructiveButtonTitle:nil otherButtonTitles:@"Purchase", @"Preauth",
nil];
    [transTypeSheet showInView:self.view];
}

-(void)actionSheet:(UIActionSheet *)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex
{
    self.selectedIdx = buttonIndex;

    if (buttonIndex < [transTypes count])
    {
        [self.btnTransType setTitle:[transTypes objectAtIndex:buttonIndex]
forState:UIControlStateNormal];
        if ([[transTypes objectAtIndex:buttonIndex] isEqualToString:@"Purchase"])
        {
            [self.btnRecur setEnabled:YES];
        }
        else
        {
            [self.btnRecur setEnabled:NO];
        }
    }
}

-(void)paymentAuthorizationViewControllerDidFinish:(PKPaymentAuthorizationViewController *)controller
{
    NSLog(@"payment finished!!!");
    [controller dismissViewControllerAnimated:YES completion:nil];
}

-(void)recurEnabled:(BOOL)enabled startNow:(BOOL)startNow numRecur:(NSInteger)numRecur
amount:(NSString *)amount startDate:(NSString *)date unit:(NSString *)unit
interval:(NSInteger)interval
{
    self.hasRecur = enabled;
    self.configured = YES;

    if (self.hasRecur)
    {
        self.recurAmount = amount;
        self.recurStartDate = date;
        self.recurStartNow = (startNow)? @"true" : @"false";
        self.recurUnit = unit;
        self.numOfRecurs = numRecur;
        self.recurInterval = interval;
    }
    [self updateGui];
}
@end
```

6. How do I Test My Solution?

Test credentials

When testing, your code should use the following **test credentials**:

```
MpgRequest *req = [MpgRequest mpgRequestWithStoreId:@"moneris" ApiToken:@"hurgle"
Transaction:transaction];
[req setTestMode:YES];
[req setProcCountry:@"CA"];
```

Simulator host:

The test environment has been designed to replicate our production environment as closely as possible. One major difference is that we are unable to send test transactions onto the production authorization network and thus Issuer responses are simulated. *The test environment will approve and decline transactions based on the penny value of the amount sent.* For example, a transaction made for the amount of $9.00 or $1.00 will approve since the .00 penny value is set to approve in the test environment. Transactions in the test environment should not exceed $11.00. For a list of all current test environment responses for various penny values, please see the **Test Environment Penny Response Table** available at https://developer.moneris.com.

These responses may change without notice. Moneris Solutions recommends you regularly refer to our website to check for possible changes.

7. How Do I Configure My Store For Production?

Once you have completed your testing you are ready to point your store to the production host. You will need to change the Test Mode to NO. You will also be required to change the store Id and API Token.

PRODUCTION	DEVELOPMENT

`MpgRequest*req=[MpgRequest mpgRequestWithStoreId:@"xxxxxx"ApiToken:@"xxxxx" Transaction:transaction];`
`[req setTestMode:NO];`
`[req setProcCountry:@"CA"];`
`initWithHost:@"www3.moneris.com"`
	`MpgRequest*req=[MpgRequest mpgRequestWithStoreId:@"moneris"ApiToken:@"hurgle" Transaction:transaction];`
	`[req setTestMode:YES];`
	`[req setProcCountry:@"CA"];`
	`initWithHost:@"esqa.moneris.com"`

8. Moving into Production.

When moving your merchant/account to production you will be required to configure/pass transaction credentials assigned to your merchant/account by Moneris. The credentials known as Store Id and API Token are assigned when merchants complete the store activation process as described in the Welcome letter. In cases where you are enabling multiple physical locations for a given merchant/account, it is important that this is a coordinated effort between you and/or your merchant customer in order to minimize input errors and to ensure that the correct payment credentials are assigned to each Moneris Gateway account/store.

By default batch-close is set to occur automatically on a daily basis between 10 and 11 pm Eastern Time. If your system requires the merchant account's batch-close to be controlled by your system, please specify this requirement in your certification test results. In addition, your product documentation should describe the batch-close processes and instruct merchants to contact Moneris to have their batch-close set to manual.

9. How Do I Get Help?

	Start	Development	Production
When do you need us	*Just getting started and need an integration specialist?*	*Already working with an integration specialist and need development assistance?*	*Already have a live application and need production support?*
Who we are	**Client Integration Specialists**	**eProducts Technical Consultants**	**Customer Service**
What we do	Integration & Certification Support	Development & Technical Support	Financial & Technical Merchant Support
How to reach us	E: ClientIntegrations@moneris.com Hours: Monday – Friday 8:30am to 8pm EST	Phone: 1-866-562-4354 Email: eproducts@moneris.com Hours: Monday – Friday 8am to 8pm EST	Phone: 1-866-319-7450 Hours: 7/24

10. Appendix A: MpgTransactionRequest Properties

Request Parameters			
Variable Name	**Size/Type**	**Description**	**Class Properties**
StoreId	10 / an	This is defined and provided by Moneris Solutions and is used to identify the merchant.	NSString storeId
ApiToken	20 / an	This is defined and provided by Moneris Solutions and is used in conjunction with the Store ID to uniquely identify your store.	NSString apiToken

Host	An	Development = esqa.moneris.com
		Production = www3.moneris.com
			NSString host

11. Appendix B: Definition of Request Fields

Request Fields		
Variable Name	**Size/Type**	**Description**
orderId	50 / an (variable length)	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 / 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 / an	Street Number & Street Name (max – 19 digit limit for street number and street name combined). This must match the address that the issuing bank has on file.
avs_zipcode	10 / an	Zip or Postal Code – This must match what the issuing banks has on file.
cvd_value	4 / num	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 / num	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.

CustInfo Request Field

Field Name	Size/Type	Description
Shipping Information		
first name	30 / an	
last name	30 / an	
company name	50 / an	
address	70 / an	
city	30 / an	
province	30 / an	
postal code	30 / an	
country	30 / an	
phone	30 / an	
fax	30 / an	
tax1	10 / an	
tax2	10 / an	
tax3	10 / an	
shipping_cost	10 / an	
Extra Details		
email	60 / an	
instructions	100 / an	

Recur Request Field

Field Name	Size/Type	Description
recur_unit	day, week, month, eom	The unit that you wish to use as a basis for the Interval. This can be set as day, week, month or end of month. Then using the “period” field you can configure how many days, weeks, months between billing cycles.
period	0 – 999 / num	This is the number of recur_units you wish to pass between billing cycles. Example : period = 45, recur_unit=day -> Card will be billed every 45 days. period = 4, recur_unit=weeks -> Card will be billed every 4 weeks. period = 3, recur_unit=month -> Card will be billed every 3 months.

		period = 3, recur_unit=eom -> Card will be billed every 3 months (on the last day of the month) Please note that the total duration of the recurring billing transaction should not exceed 5-10 years in the future.
start_date	YYYY/MM/DD	This is the date on which the first charge will be billed. The value must be in the future. It cannot be the day on which the transaction is being sent. If the transaction is to be billed immediately the start_now feature must be set to true and the start_date should be set at the desired interval after today.
start_now	true / false	When a charge is to be made against the card immediately start_now should be set to ‘true’. If the billing is to start in the future then this value is to be set to ‘false’. When start_now is set to ‘true’ the amount to be billed immediately may differ from the recur amount billed on a regular basis thereafter.
recur_amount	9 / decimal	Amount of the recurring transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99. This is the amount that will be billed on the start_date and every interval thereafter.
num_recurs	1 – 99 / num	The number of times to recur the transaction.
amount	9 / decimal	When start_now is set to ‘true’ the amount field in the transaction array becomes the amount to be billed immediately. When start_now is set to ‘false’ the amount field in the transaction array should be the same as the recur_amount field.

12. Appendix C: Definition of Response Variables

Response Fields		
Variable Name	**Size/Type**	**Description**
ReceiptId	50 / an	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 / num	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. **Format**: nnnnnnnnnnnnnnnnnn (18 digits)
ReponseCode	3 / num	Transaction Response Code < 50: Transaction approved >= 50: Transaction declined NULL: Transaction was not sent for authorization Custom Apple Pay SDK Responses: ResponseCode / Message / Definition 900 / Global Error / Unable to decrypt payload * If you would like further details on the response codes that are returned please see the Response Codes document available at https://developer.moneris.com
ISO	2 / num	ISO response code
AuthCode	8 / an	Authorization code returned from the issuing institution

TransTime	##:##:##	Processing host time stamp. Time of the transaction. Must be displayed on the transaction receipt.
		Format: HH:II:SS
		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. **Format**: YYYY-MM-DD YYYY = 4 digit year MM = 2 digit month (“0” left padded | Jan = 01) DD = 2 digit day of month (“0” left padded)
TransType	an	Type of transaction that was performed
Complete	true/false	Transaction was sent to authorization host and a response was received
Message	100 / an	Response description returned from issuing institution.

TransAmount	9 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. **Format**: nnnnnnN.NN 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
CardType	2 / alpha	Credit Card Type
Txn_number	20 / an	Gateway Transaction identifier
TimedOut	true/false	Transaction failed due to a process timing out
Ticket	n/a	reserved
CorporateCard	true/false	Indicates whether the card is a corporate card or not.
CAVV	40 / an	Decrypted CAVV value for the transaction. Returned for Apple Pay Purchase/Pre-Authorization transaction if payload is successfully decrypted.
DeviceManufacturerIdentifier	12 / an	Token requestor ID. Returned from decrypted payload. Hex-encoded device manufacturer identifier.
RecurSucess	true/false	Indicates whether the transaction successfully registered.
AvsResultCode	1 / alpha	Indicates the address verification result.
CvdResultCode	2 / an	Indicates the CVD validation result.

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

13. Appendix D: Error Messages

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. A 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, either because they are offline or you are unable to connect to the internet. A 'NULL' can also be returned when a transaction message is improperly formatted.

Below are error messages that are returned in the Message field of the response.

Message: XML Parse Error in Request: <System specific detail>
Cause: For some reason an improper XML document was sent from the API to the servlet

Message: XML Parse Error in Response: <System specific detail>
Cause: For some reason an improper XML document was sent back from the servlet

Message: Transaction Not Completed Timed Out
Cause: Transaction times out before the host responds to the gateway

Message: Request was not allowed at this time

Cause: The host is disconnected

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

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

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

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

Custom Apple Pay SDK Responses:

ResponseCode	Message	Definition
900	Global Error	Unable to decrypt payload

14. Appendix E: Security Requirements

It is important to note that 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 2.0 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) 2.0. 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 http://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 download the **PCI-DSS Implementation Guide**.