



Visa Checkout

SDK for iOS

Effective: September 13, 2016



Important Note on Confidentiality and Copyright

The Visa Confidential label signifies that the information in this document is confidential and proprietary to Visa and is intended for use only by Visa Clients subject to the confidentiality restrictions in Visa's Operating Regulations, non-Client Third Party Processors that have an executed and valid Exhibit K on file with Visa, and other third parties that have a current nondisclosure agreement (NDA) or other agreement with Visa that covers disclosure of the information contained herein. This document is protected by copyright restricting its use, copying, distribution, and decompilation. No part of this document may be reproduced in any form by any means without prior written authorization of Visa. Visa and other trademarks are registered trademarks of Visa International Service Association, and are used under license by Visa. All other product names mentioned herein are the trademarks of their respective owners.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN: THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. VISA MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Note: *All third party brand names, logos and other intellectual property contained within this document are the property of their respective owners, are used for identification or demonstrative purposes only, and do not imply product endorsement or affiliation with Visa.*

If you have technical questions or questions regarding a Visa service or capability, contact your Visa representative.

Contents

Chapter 1 • Getting Started With the Visa Checkout SDK

What's New in This Version	1-1
Downloading the Visa Checkout SDK for iOS	1-2
Visa Checkout SDK for iOS Overview	1-2
Visa Checkout Integration Options	1-3
Standard Integration	1-3
Skip Shipping Information	1-3
Visa Checkout Transaction Flow	1-3
Public and Private Key Security.	1-5
Visa Checkout SDK for iOS Information Collection.	1-5

Chapter 2 • Setting Up Your Xcode Environment

Standard Setup	2-1
Additional Instructions for Swift Coding	2-3

Chapter 3 • Setting Up the View Controller

Prerequisites	3-1
Steps	3-1
Launching the SDK From a Custom View	3-8

Displaying Card Art	3-8
Determining Whether a Device Has Been Jailbroken	3-9

Chapter 4 • Visa Checkout Library Delegate Methods

checkoutSuccessWithSummary	4-1
checkoutFailedWithError	4-2
checkoutCancelled	4-3
VisaPaymentInfo	4-4
VisaServer	4-10
GAISharedInstance	4-11

Chapter 5 • Update Payment Info Pixel Image

Update Payment Info Pixel Image Summary	5-1
Update Payment Info Pixel Image Request	5-2
Update Payment Info Pixel Image Response	5-5
Update Payment Data Info Pixel Image Error Messages	5-5
Update Payment Info Pixel Image Examples	5-6

Chapter 6 • Update Payment Info

Update Payment Info Summary	6-1
Update Payment Info Request	6-2
Update Payment Info Errors	6-10
Update Payment Info Examples	6-11
Order Update Success Example	6-11
Payment Update Example	6-12
Update Payment Info Error Examples	6-14
Update Payment Info Error Example	6-14
Update Multiple Info Structure Examples	6-15

Chapter 7 • Consumer Information

User Data Properties	7-2
Payment Request Properties.	7-4
Payment Instrument Properties.	7-6
Token Info Properties	7-8
Cryptogram Info Properties	7-9
Payment Type Properties	7-10
Address	7-10
Card Arts.	7-14
Expiration Date	7-15
Risk Properties.	7-15
3-D Secure Authentication Data Fields	7-16

Appendix A • Decrypting Consumer Information

Consumer Information Decryption Algorithm	A-1
Consumer Information Decryption Examples.	A-2

Appendix B • SHA256–Bit Hashing

SHA256–Bit Hashing Algorithm	B-1
Creating a Test x-pay-token Header	B-1
SHA256–Bit Hashing Examples	B-3

Appendix C • AVS and CVV Responses

AVS Codes	C-1
CVV Codes	C-2

Appendix D • US, Canadian, and Australian Location Abbreviations

United States Abbreviations for States and Mailing Locations.	D-1
Canadian Province Abbreviations	D-3

Australian State and Territory Abbreviations	D-3
--	-----

Appendix E • SDK Upgrade and Discontinuance Information

Appendix F • What’s New in Prior Releases

What’s New in Version 4.0	F-1
What’s New in Version 3.7	F-1
What’s New in Version 3.5	F-2

Appendix G • Document Revision History

Getting Started With the Visa Checkout SDK

1

This manual is intended for experienced programmers writing Objective C code for iOS apps. Currently, the following devices are supported: iPhone 5 or later running iOS 8.0 or later.

What's New in This Version

- Visa Checkout billing and shipping are supported for the following additional countries:
 - France
 - Ireland
 - Poland
 - Spain
 - United Kingdom
- The **Pay** button in the lightbox can be used with any currency supported by Visa Checkout.
- If you are a merchant in United Kingdom, France, Spain, Ireland, or Poland, you must allow issuers to authenticate consumers from these countries on the first use of a card by the consumer in Visa Checkout. Contact your Visa Checkout representative to enable and configure Verified by Visa for Visa cards and SecureCode for Mastercard cards to perform authentication when a consumer's card is first used. If you fail to enable Verified by Visa and SecureCard for this purpose, you are responsible for allowing the issuer to authenticate the consumer on a card's first use. If you accept American Express cards from these countries, you must allow the authentication of the consumer outside of Visa Checkout on every transaction; American Express Safekey support is not currently provided by Visa Checkout. For more information, see the *Visa Checkout Integration Guide*.
- Compliance with European Union privacy requirements is provided through the display of a cookie/privacy banner with a link to the privacy policy and the option to allow consumers to opt-out of non-essential cookies and consumer emails. The privacy policy and ability to opt-out of non-essential cookies is available to all Visa Checkout users.

- During enrollment of a Polish consumer, an additional consent check box appears to support mandated international sanctions verification.

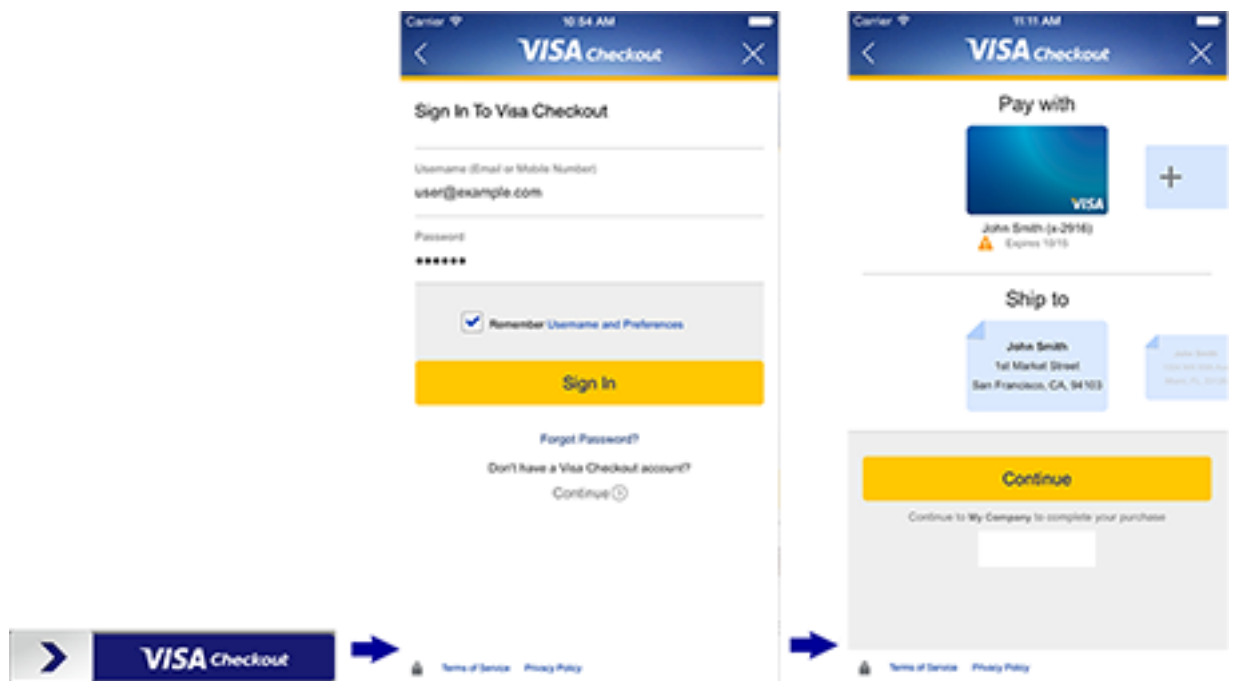
Downloading the Visa Checkout SDK for iOS

You can download the Visa Checkout SDK for iOS from the [Resources folder](https://developer.visacheckout.com/documentation) at <https://developer.visacheckout.com/documentation>.

Visa Checkout SDK for iOS Overview

The Visa Checkout SDK for iOS enables you to integrate a Visa Checkout payment flow into your iOS app. The consumer taps the checkout with Visa Checkout button in your app, which launches the Visa Checkout flow. The consumer then logs into Visa Checkout to confirm the payment information and shipping address and continue with the purchase, possibly changing shipping or payment information before continuing back to your app. At that point, your app confirms a the successful purchase with Visa Checkout.

After following the integration steps described in this document, your customer can checkout and pay with Visa Checkout:



When the customer taps the **Visa Checkout** button, Visa Checkout executes the following flow:

1. Visa Checkout displays the Sign In screen, which allows your customer to
 - Sign in by entering an e-mail address or mobile phone number, and a password.
 - Create a Visa Checkout account and specify payment information and shipping details to be securely stored in Visa Checkout.
2. Visa Checkout presents the Review and Continue screen.

Note: At this point, the customer can add or change the payment method, billing address, and shipping address.

3. After the customer taps **Continue**, Visa Checkout returns the call ID, which identifies the payment request, an encrypted payload and other information, then returns control to your app. For additional information, see [checkoutSuccessWithSummary](#).

After control is returned to your app, you pass the encrypted payment payload to your server for authorization and capture. Once done, your server must update the payment information in Visa Checkout

Visa Checkout Integration Options

The Visa Checkout mobile payment flow supports several different integration options in order to provide the consumers with a streamlined experience. These options include standard integration and suppressing shipping information.

Standard Integration

In a standard integration, you specify all required and optional fields to be displayed:

- Countries for which shipping is available
- Accepted payment methods
- Your merchant logo or name
- An amount and associated currency
- Information about the payment instrument

This enables a consumer to login and only have to confirm or change this information.

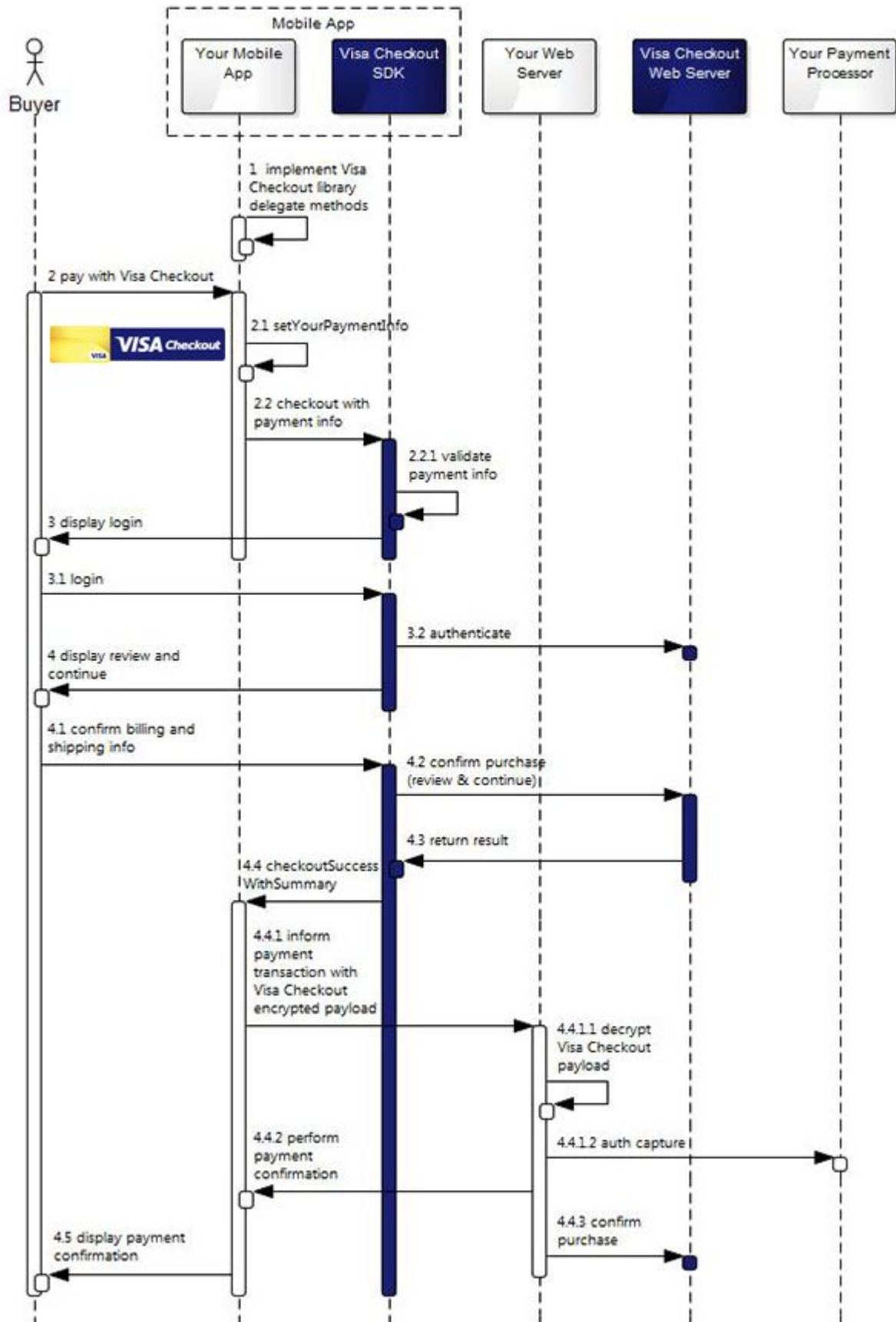
Note: For a complete list of fields, see [VisaPaymentInfo](#).

Skip Shipping Information

In an integration that does not require shipping information, such as a payment for a download, you specify all required and optional fields except the shipping address and you specify that shipping is not required, which causes the shipping address panel to not be displayed.

Visa Checkout Transaction Flow

The following flow diagram shows how the Visa Checkout SDK for iOS interacts with a merchant app and backend systems:



The diagram depicts properly implemented `VisaCheckoutFramework` delegate methods in your mobile app. These methods enable the app to configure the Visa Checkout SDK for iOS and receive status information with the transaction completes. The remaining steps show the interaction between the app and Visa Checkout

1. When the consumer taps the **Visa Checkout** button, your app initializes the payment information, using a delegate method, to pass critical information such as the amount and currency.
2. The consumer signs in or creates a Visa Checkout account.
3. Visa Checkout returns an encrypted payload and key.
4. The app sends the payload and key to your server, which initiates Visa Checkout actions.

For information about back-end actions required by your server, see *Visa Checkout Integration Guide*.

You should only initialize the SDK when the button is about to shown on the screen, preferably in the same view controller.

Important:

Don't initialize the SDK multiple times.

Public and Private Key Security

If you are calling the Visa Checkout API, you will need to follow these rules to implement essential security controls:

- At a high-level, the security of server communications is provided by the use of public-private key pairs. You communicate to Visa Checkout with your *API key*. You include a *shared secret*, along with data that needs to be protected from tampering, all of which are encrypted using an SHA256-bit hashing algorithm.

Important:

You must only use a key and shared secret provided to you by Visa; specifically, you cannot communicate with Visa Checkout using keys or secrets that are not authorized for your use.

- YOU ARE SOLELY RESPONSIBLE FOR MAINTAINING ADEQUATE SECURITY AND CONTROL OF ANY API OR SHARED SECRET KEYS PROVIDED TO YOU. Because the shared secret ensures secure communications between you and Visa Checkout, you must protect the shared secret, allowing only authorized and authenticated entities, e.g. people, APIs, code, etc., to access the shared secret. The shared secret should never be stored or available unencrypted on a web page. The shared secret must be protected using either hardware- or software- based strong encryption, such as AES. In addition, you must provide your own secure server to store the encrypted shared secret. Because the shared secret is hashed along with changing data, you will need to create hash strings in real time.

Visa Checkout SDK for iOS Information Collection

The Visa Checkout SDK for iOS collects Visa Checkout enrollment and sign-in information, attributes of the device upon which it is installed, and de-identified usage information of the SDK. Visa Checkout uses and shares such information in accordance with the Visa Checkout privacy policy. The SDK may send information directly to its service providers to facilitate Visa's use and analysis of such information.

Visa recommends you consult your privacy or legal department to determine any required notices or consents in connection with your incorporation and activation of the Visa Checkout SDK for iOS.

Setting Up Your Xcode Environment 2

You must follow the steps for standard setup for iOS development. There are additional steps for Swift setup.

Standard Setup

To set up your Xcode environment:

1. Download and unzip the SDK package.

The contents include separate binaries, one for universal build and another for app store build (device-only architecture), each containing:

- `VisaCheckoutLibrary.bundle`
- `VisaCheckoutFramework.framework`
- `sample` folder containing the sample project

2. Add the SDK to your project:

- Drag `VisaCheckoutLibrary.bundle` and `VisaCheckoutFramework.framework` into your target Xcode project folder.
- Check the **Copy items into destination group's folder (if needed)** box if it is not already checked.
























3. Add dependencies.

- a. Click the **Targets** → *YourApp* → '**Build Phases**' tab.
- b. Expand '**Link Binary With Libraries**'.
- c. Click the + icon in the bottom left corner of the '**Link Binary With Libraries**' panel and add the following libraries:
 - `AddressBook.framework`
 - `AddressBookUI.framework`
 - `AdSupport.framework` (optional)
 - `AudioToolbox.framework`
 - `CoreData.framework`
 - `CoreLocation.framework`
 - `CoreTelephony.framework`

- CoreText.framework
- CoreVideo.framework
- CodeMedia.framework
- MobileCoreServices.framework
- OpenGLES.framework
- AVFoundation.framework
- libCardIO.a (see sample directory)
- libGoogleAnalyticsServices.a (see sample directory)
- Libsqlite3.dylib
- Libz.dylib
- QuartzCore.framework
- Security.framework
- SystemConfiguration.framework
- Foundation.framework
- CoreGraphics.framework
- UIKit.framework
- LocalAuthentication.framework
- FBSDKCoreKit.framework (optional)
- FBSDKShareKit.framework (optional)
- FBSDKLoginKit.framework (optional)

Your **'Link Binary With Libraries'** panel should look like the following one:

▼ Link Binary With Libraries (23 items)

Name	Status
 VisaCheckoutLibrary.framework	Required ↕
 MobileCoreServices.framework	Required ↕
 CoreMedia.framework	Required ↕
 AudioToolbox.framework	Required ↕
 AVFoundation.framework	Required ↕
 LocalAuthentication.framework	Required ↕
 AddressBook.framework	Required ↕
 AddressBookUI.framework	Required ↕
 AdSupport.framework	Required ↕
 CoreData.framework	Required ↕
 CoreLocation.framework	Required ↕
 CoreTelephony.framework	Required ↕
 CoreText.framework	Required ↕
 libsqlite3.dylib	Required ↕
 libz.dylib	Required ↕
 QuartzCore.framework	Required ↕
 Security.framework	Required ↕
 SystemConfiguration.framework	Required ↕
 CoreGraphics.framework	Required ↕
 UIKit.framework	Required ↕
 libGoogleAnalyticsServices.a	Required ↕
 Foundation.framework	Required ↕
 libCardIO.a	Required ↕

4. Select **Build Settings** → **Linking** → **Other Linker Flags** and add the following lines in **Debug & Release** mode:

```
-all_load  
-ObjC  
-lc++
```

Additional Instructions for Swift Coding

The SDK provides a .swift version of the merchant view controller in this package. It provides Swift bridged versions of Visa library protocols. To set up your Xcode environment, follow the instructions in [Standard Setup](#) first, then follow these instructions:

1. Select **Build Settings** for the application target and set the following flags, if not already set:
 - a. Set **Embedded Content Contains Swift Code** to YES
 - b. Set **Linking** → **Other Linker Flags** to `all_load`, `-ObjC`, `-lc++`, `-lsqlite3`, `-lz`
 - c. Set **Framework Search Path** to `$(PROJECT_DIR)` or to the framework to be added
 - d. Set **Library Search Path** to `$(PROJECT_DIR)/CardIO`, `$(PROJECT_DIR)/GA` or to the directory where cardio and GA libraries have been added
 - e. Set **Objective-C bridging header** to `$(PROJECT_DIR)/SwiftTest-Bridging-Header.h` or to the location of your bridging header file
2. Add a following import line in the bridging header file:

```
#import <VisaCheckoutLibrary/VisaCheckoutFramework.h>
```

Note: A bridging header file can be added, just like any other new file. Ensure it is linked to the build settings you set in Step 1e.

Setting Up the View Controller 3

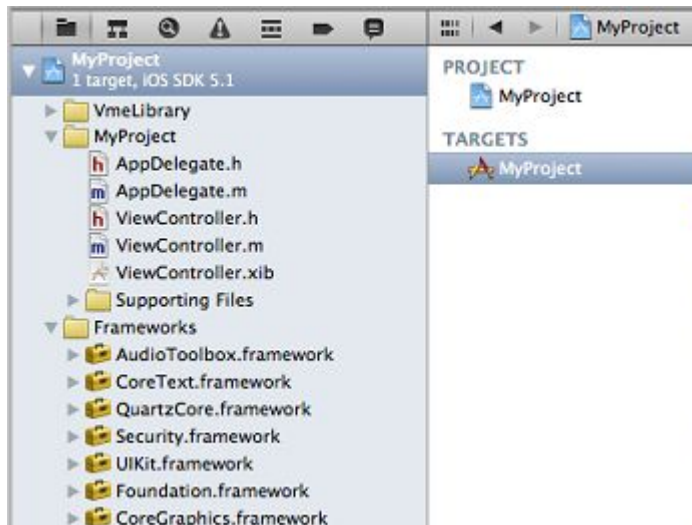
After you set up the Xcode environment, you are ready to work with the actual code. This chapter shows the minimal steps necessary to integrate the Visa Checkout SDK for iOS into your project

Important:

You cannot debug the SDK on a device running against the production environment. Attempting to do so will result in a 500 error.

Prerequisites

Your project should contain a view controller, which you will modify to include the Visa Checkout button and the delegate methods that control the presentation of the Visa Checkout flow in your app:



Steps

In the view controller that displays the Visa Checkout button and responds to clicks on the button:

1. Import `VisaCheckoutFramework.h` into your view controller and adopt the `VisaLibraryDelegate` protocol.

```
#import <UIKit/UIKit.h>
#import <VisaCheckoutFramework/VisaCheckoutFramework.h>

@interface ViewController : UIViewController <VisaLibraryDelegate>
@end
```

Important:

You must implement all required methods in `VisaLibraryDelegate.h`.

2. Add the Visa Checkout button to your view.

There are 2 variations of the button, a standard button and a neutral button. You should use the standard button unless the background colors in your app are dark, in which case, you can consider using a neutral button to provide greater contrast.

You can add the Visa Checkout button in one of two ways, either through code or through the storyboard view.

Note: *The frame width determines the actual size of a Visa Checkout button, whose height is sized to retain the original aspect ratio. The frame width must be between 154 points and 213 points, inclusive. The origin (x and y coordinates) you specify position the button at the desired location.*

For an express checkout button, the width must be between 213 pixels, up to the screen width. The height is 77 pixels.

- **Code method:**

If you are using a Visa Checkout button, add the following button code to your `loadView` or `viewDidLoad` method:

```
VisaCheckoutButton *checkoutButton =
    [[VisaCheckoutButton alloc] initWithFrame: CGRectMake(
        30, 80, 0, 0)];
checkoutButton.delegate = self;
// Set the button color to neutral, if necessary
// self.checkoutButton.color = VisaCheckoutButtonColorNeutral;
[self.view addSubview:checkoutButton];
```

If you are using an express checkout button, add the following button code to your `loadView` or `viewDidLoad` method:

```
//Add the following button code to your loadView or viewDidLoad
VisaExpressCheckoutView *exoButtonContainer =
    [[VisaExpressCheckoutView alloc] initWithFrame: CGRectMake(
        0, 0, 200, 77)];
exoButtonContainer.expressCheckoutdelegate = self;
// Set the button color to neutral, if necessary
exoButtonContainer.color = VisaCheckoutButtonColorNeutral;
[self.view addSubview:exoButtonContainer];
```

- **Storyboard method:**

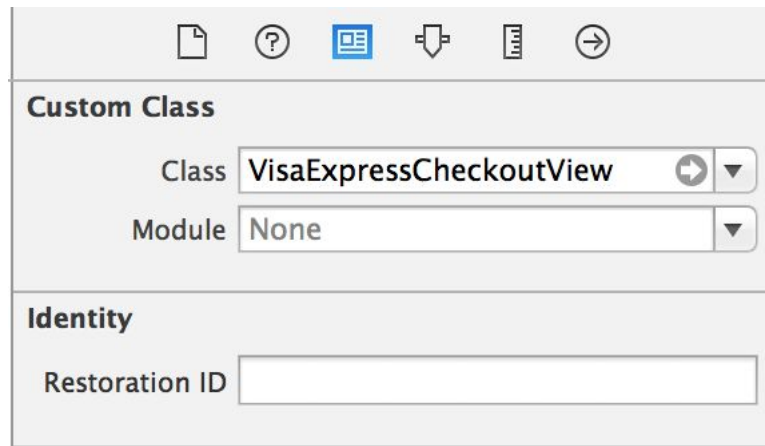
1. Add a `UIButton` to your storyboard's view.

Note: *You must remove or clear the button title.*

The following item is an example of an express checkout button:

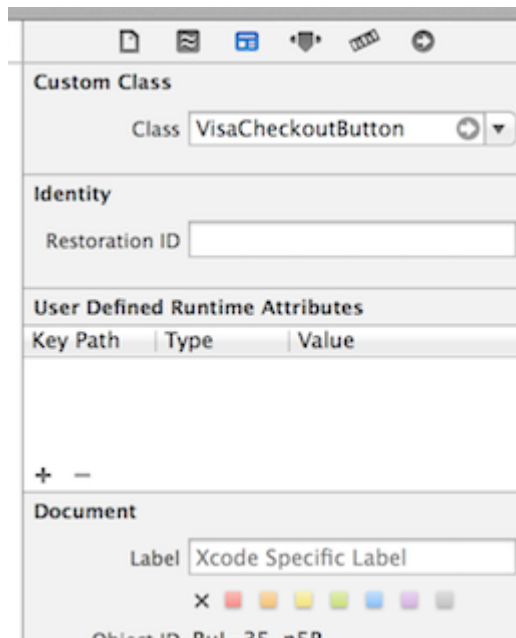
Default State**Email Sign In****After Click****Password Entry**

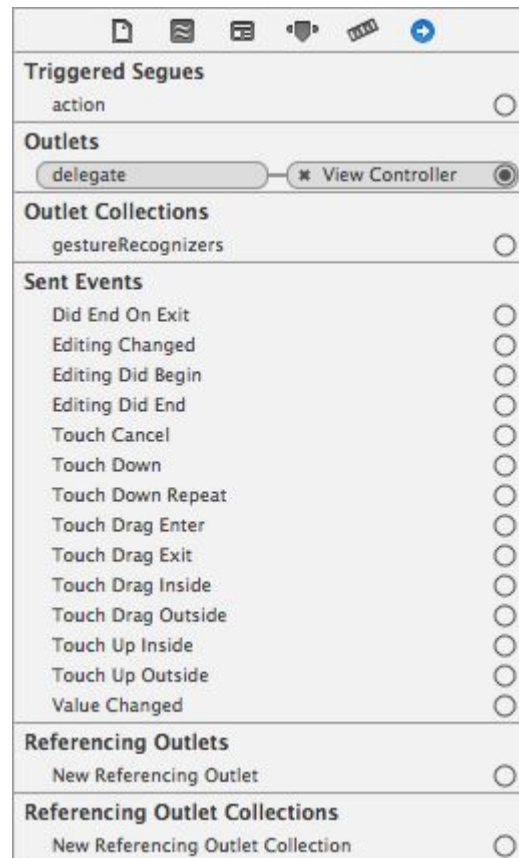
2. Specify its class as either `VisaCheckoutButton` or `VisaExpressCheckoutView`, depending on the kind of button:



3. Assign your controller to the delegate outlet.

The following screenshots show the configuration for a Visa Checkout Button:





3. Implement the `VisaLibraryDelegate` methods for the Visa Checkout Sandbox or production servers. For more information, see *Visa Checkout Library Delegate Methods*. The substeps identify only a few of the methods:
 - a. Implement your `VisaServer` delegate method to identify whether the endpoint is to a Visa Checkout Sandbox server or a Visa Checkout production server.

You specify the server with one of the following constants:

- `kVisaSandboxServer` specifies the Sandbox server. The API key and shared secret key are available from <https://checkout.visa.com/business/index.jsp>.
- `kVisaProductionServer` specifies a production server. The API key and shared secret key are available from your Visa representative.

The following example shows an implementation of the `VisaServer` delegate method:

```
- (VisaServer)visaServer {
    return kVisaSandboxServer;
}
```

- b. Implement your `VisaPaymentInfo` delegate method to initialize the payment attributes.

The following example shows an integration. Your integration should include the values such as the amount of the transaction, the currency, and such:

Note: For amounts of currency, the amount is restricted to a maximum of 9 digits and 2 digits of precision. Strings, such as the display name and description, generally should be less than 100 characters.

```
- (VisaPaymentInfo *)paymentInfo {

    VisaPaymentInfo *paymentInfo = [[VisaPaymentInfo alloc] init];

    // Uncomment the following lines to pass amount
    [paymentInfo setAmount:[NSDecimalNumber
decimalNumberWithString:@"9.99"]];
    [paymentInfo setCurrency::VCurrencyCodeUS];
    paymentInfo.merchantRequestId = @"a001";
    paymentInfo.subtotal = [NSDecimalNumber decimalNumberWithString:@"0"];
    paymentInfo.shippingHandlingCharges =
        [NSDecimalNumber decimalNumberWithString:@"0"];
    paymentInfo.tax = [NSDecimalNumber decimalNumberWithString:@"0"];
    paymentInfo.discount = [NSDecimalNumber decimalNumberWithString:@"0"];
    paymentInfo.giftWrapCharges = [NSDecimalNumber decimalNumberWithString:@"0"];
    paymentInfo.miscCharges = [NSDecimalNumber decimalNumberWithString:@"0"];
    paymentInfo.paymentDescription = @"sample transaction";
    paymentInfo.orderId = @"111";
    paymentInfo.promoCode = @"freeCheckout";
    paymentInfo.isShippingAddressRequired = YES;
    // SUMMARY, FULL (includes account number), or NONE (only call ID)
    paymentInfo.datalevel = @"SUMMARY";

    // Use an existing call ID from an earlier session
    // to directly launch the sign-in flow with the card and addresses
    // set in the referenced call ID.
    paymentInfo.setReferenceCallId = "...";
    // Pass merchant information
    VisaMerchantInfo *merchantInfo = [[VisaMerchantInfo alloc] init];
    // Uncomment the next line after filling in the API Key
    // merchantInfo.merchantAPIKey = [replace with API key];
    merchantInfo.userReviewAction = kVisaUserReviewActionContinue;
    merchantInfo.displayName = @"My Merchant Name";
    merchantInfo.externalProfileId = @"default";
    merchantInfo.displayName = @"The Merchant App";
    merchantInfo.customReviewActionContinueMessage = @"Add custom message here";
    // Uncomment the following line after ensuring that the (mercLogo) asset is added
    // to the project:
    // merchantInfo.merchantLogoAssetName = @"mercLogo";

    merchantInfo.canadianDebitCardSupport = VAcceptCanadianDebitCard;

    // Choose a preferred locale when SDK is launched
    merchantInfo.preferredLocale = VLocale_enUS;

    paymentInfo.merchantInfo = merchantInfo;

    // Set up Verified by Visa (3D-Secure)
    VisaThreeDSSetUp *threeDSSetUp = [VisaThreeDSSetUp new];
    threeDSSetUp.threeDSActive = VisaActivate3DS;
    paymentInfo.threeDSSetUp = threeDSSetUp;

    NSMutableArray *cardBrandArray =
```

```

[[NSMutableArray alloc] initWithObjects:@"Visa", @"Mastercard", @"AMEX",
    @"Discover", nil];
paymentInfo.acceptedPaymentTypesAndBrands = cardBrandArray;

// Set accepted countries for shipping

NSMutableArray *acceptedCountryList =
    [[NSMutableArray alloc] initWithObjects:@"US",@"CA", @"AU", nil];
paymentInfo.acceptedShippingCountries = acceptedCountryList;

// Set accepted countries for billing
NSMutableArray *acceptedBillingCountryList =
    [[NSMutableArray alloc] initWithObjects:@"CA", @"US", @"AU" , nil];
paymentInfo.acceptedBillingCountries = acceptedBillingCountryList;

// Specify an external client ID if needed
// merchantInfo.externalClientId = @"MyClientID";

// Set custom data (optional)
paymentInfo.merchantInfo.customData = @{
    @"nvPair" : @[
        @{
            @"name" : @"customName1",
            @"value" : @"customValue1"
        },
        @{
            @"name" : @"customName2",
            @"value" : @"customValue2"
        }
    ]};

// Enable tokenization
paymentInfo.enableTokenization = VEnableTokenization;

...
return paymentInfo;
}

```

4. Implement your `checkoutSuccessWithSummary` delegate method to return the call ID from your app to your server. Your server communicates with the Visa Checkout server.

The following example shows an implementation of the `checkoutSuccessWithSummary` delegate method:

```
- (void)checkoutSuccessWithSummary:(VisaPaymentSummary*)paymentSummary {
    ...

    /* Handle request with the following response fields:
       paymentSummary.callId;
       paymentSummary.lastFourDigits;
       paymentSummary.cardBrand;
       paymentSummary.cardType;
       paymentSummary.countryCode;
       paymentSummary.postalCode;
       paymentSummary.encKey;
       paymentSummary.encPaymentData;
    */
    ...
}
```

5. Take the response fields from the previous step and pass it on to your server based code. Modify your server-based code to update the purchase.

The `checkoutSuccessWithSummary` method returns a payload that contains the consumer's information, which is encrypted. Before you decrypt the payload, you must move it (both its key, `encKey`, and the encrypted data, `encPaymentData`), to a secure server. You must determine the part of the payload that you need; for example, you may need complete shipping information. For the contents of the payload see [Consumer Information](#).

Note: After you finish, you must update the payment information in Visa Checkout if your processor does not handle it. For more information, see [Update Payment Info](#).

Launching the SDK From a Custom View

If a custom view is used, the view will either have an `IBAction` or a `gestureRecognizer` method. In this case, you can launch the SDK from within your method, as follows:

```
self.visaCheckout = [[VisaCheckout alloc] init];
[self.visaCheckout invokeVisaCheckout:^(NSError *error) {

    if (error) {
        NSLog(@"Error Details...%@", [error localizedDescription]);
    }

} withOptions:nil withDelegate:self];
```

Note: When you use the Visa Checkout button, launching of the SDK is performed automatically.

Displaying Card Art

You can display the card art by including the following lines in your view controller; for example, in the `viewDidLoad` method:

```
self.visaCheckoutObject = [[VisaCheckout alloc] init];  
[self.paymentMarkCardArt setImage:[self.visaCheckoutObject fetchCardArtImage]];
```

The recommended card art size is 49 pixels wide by 31 pixels high. It is your view controller's responsibility to fetch and update the card art whenever necessary; for example, the view controller might check to determine whether the card with associated art changed after a transaction completes.

Determining Whether a Device Has Been Jailbroken

Visa Checkout is not supported on jailbroken devices. To enable detection of jailbroken devices, include `cydia` in the `LSApplicationQueriesSchemes` section of the plist:

```
<key>LSApplicationQueriesSchemes</key>  
  <array>  
    <string>cydia</string>  
  </array>
```


Visa Checkout Library Delegate Methods

4

The Visa Checkout library calls delegate methods when the Visa Checkout SDK for iOS begins to relinquish control. The events correspond to the consumer completing the checkout flow, the consumer canceling the checkout, an exception raised during the checkout flow, and completion of exiting animations.

checkoutSuccessWithSummary

Visa Checkout calls the `checkoutSuccessWithSummary` method when a customer successfully checks out. Specify actions to initiate processing of the transaction in this method.

Syntax

```
- (void)checkoutSuccessWithSummary:(VisaPaymentSummary*)paymentSummary { ... }
```

Input Parameters

None

Output Parameters

Parameter	Description
paymentSummary	The response from Visa Checkout.

The `VisaPaymentSummary` object is defined as follows:

```
@interface VisaPaymentSummary : NSObject

@property (nonatomic, strong) NSString *callId;           // call ID

@property (nonatomic, strong) NSString *encPaymentData; // encrypted payload
@property (nonatomic, strong) NSString *encKey;          // encrypted key

@property (nonatomic, strong) NSString *lastFourDigits; // last 4 digits of card
@property (nonatomic, strong) NSString *cardBrand;      // card brand, such as VISA
@property (nonatomic, strong) NSString *cardType;       // card type, such as CREDIT
@property (nonatomic, strong) NSString *countryCode;    // country code, such as US
@property (nonatomic, strong) NSString *postalCode;     // ZIP or postal code

@end
```

Return Value

None

Example

```
- (void)checkoutSuccessWithSummary:(VisaPaymentSummary*)paymentSummary {
    NSLog(@"success");

    NSString *successResponse = [NSString stringWithFormat:@" callId: %@
    \r lastFourDigits: %@
    \r cardBrand: %@
    \r cardType: %@
    \r countryCode: %@
    \r postalCode: %@
    \r encKey: %@
    \r encPaymentData: %@",
    paymentSummary.callId,
    paymentSummary.lastFourDigits,
    paymentSummary.cardBrand,
    paymentSummary.cardType,
    paymentSummary.countryCode,
    paymentSummary.postalCode,
    paymentSummary.encKey,
    paymentSummary.encPaymentData];
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Payment successful!"
    message:successResponse
    delegate:nil
    cancelButtonTitle:@"OK"
    otherButtonTitles: nil];
    [alertView show];
}
```

checkoutFailedWithError

Visa Checkout calls the `checkoutFailedWithError` method when a Visa Checkout error occurs that has not been handled by the Visa Checkout SDK for iOS. Specify actions to handle unanticipated situations; for example, logging information to your server for later analysis.

Visa Checkout calls `checkoutFailedWithError` for the following errors:

- 401 - authentication error, such as an invalid API key
- 500 - SDK running in debug mode
- 505 - outdated SDK version

Syntax

```
- (void)checkoutFailedWithError:(NSError*)error { ... }
```

Input Parameters

None

Output Parameters

Parameter	Description
error	An NSError object that contains information about an error.

Return Value

None

Example

```
- (void)checkoutFailedWithError:(NSError*)error{
    NSLog(@"Error - %@", [error description]);

    if ([error code] == 505) {
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:nil message:@"...."
            delegate:nil cancelButtonTitle:@"OK" otherButtonTitles: nil];
        [alertView show];
    }

    if ([error code] == 506) {
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:nil message:[error
            localizedDescription] delegate:nil cancelButtonTitle:@"OK" otherButtonTitles: nil];
        [alertView show];
    }
}
```

checkoutCancelled

Visa Checkout calls the `checkoutCancelled` method when a customer cancels a checkout by closing the Visa Checkout SDK for iOS, either by tapping the **Cancel** button or the grey bar at the top of the view. Specify actions to handle the cancellation; for example, returning the customer to the screen that contains the checkout button.

Syntax

```
- (void)checkoutCancelled { ... }
```

Input Parameters

None

Output Parameters

None

Return Value

None

Example

```
(void)checkoutCancelled {  
    ...  
}
```

VisaPaymentInfo

Visa Checkout calls the `VisaPaymentInfo` method when a customer taps the Visa Checkout button to obtain details of the payment from your app. For amounts of currency, the amount is restricted to a maximum of 9 digits and 2 digits of precision. Strings, such as the display name and description, generally should be less than 100 characters.

Important:

If you do not specify a card brand, shipping region, or billing region either here or in your profile, all supported brands and regions will be enabled. Set these fields or your profile to restrict brands or regions.

Note: *VisaUserInfo is not currently supported within VisaPaymentInfo.*

Payment information properties are defined as follows:

```
@class VisaMerchantInfo;  
@class VisaUserInfo; // Not currently supported  
  
@interface VisaPaymentInfo : NSObject  
  
/**  
 * Set the final amount of the transaction  
 */  
@property (strong, nonatomic) NSDecimalNumber *amount;  
  
/**  
 * Set the currency type of the transaction (ex. 'USD')  
 */  
@property (strong, nonatomic) NSString *currency;  
  
/**  
 * Set whether shipping physical goods or in store pick-up  
 */  
@property (nonatomic, assign) BOOL isShippingAddressRequired;  
  
/**
```

```

    * Set the merchant information for this checkout
    */
@property (nonatomic, strong) VisaMerchantInfo *merchantInfo;

/**
    * Set the accepted countries for shipping
    */
@property (nonatomic, strong) NSMutableArray *acceptedShippingCountries;

/**
    * Set the accepted countries for billing
    */
@property (nonatomic, strong) NSMutableArray *acceptedBillingCountries;

/**
    * Set the accepted card types and brands
    */
@property (nonatomic, strong) NSMutableArray *acceptedPaymentTypesAndBrands;

/**
    * Set the data level
    */
@property (nonatomic, strong) NSString *datalevel;

/**
    * Merchant's ID associated with the request.
    */
@property (nonatomic, strong) NSString *merchantRequestId;

/**
    * Source ID associated with the request.
    */
@property (nonatomic, strong) NSString *sourceId;

/**
    * Subtotal of the payment.
    */
@property (strong, nonatomic) NSDecimalNumber *subtotal;

/**
    * Total of shipping and handling charges in the payment.
    */
@property (nonatomic, strong) NSDecimalNumber *shippingHandlingCharges;

/**
    * Total tax-related charges in the payment.
    */
@property (nonatomic, strong) NSDecimalNumber *tax;

/**
    * Total of discounts related to the payment.
    */
@property (nonatomic, strong) NSDecimalNumber *discount;

/**
    * Total gift-wrapping charges in the payment.
    */

```

```
@property (nonatomic, strong) NSDecimalNumber *giftWrapCharges;

/**
 * Total uncategorized charges in the payment.
 */
@property (nonatomic, strong) NSDecimalNumber *miscCharges;

/**
 * payment description.
 */
@property (nonatomic, strong) NSString *description;

/**
 * Merchant's order ID associated with the payment.
 */
@property (nonatomic, strong) NSString *orderId;

/**
 * Promotion code associated with the payment.
 */
@property (nonatomic, strong) NSString *promoCode;

/**
 * CallID that can be passed in to edit payment information.
 */
@property (nonatomic, strong) NSString *referenceCallID;

/**
 * Merchants, at the time of the checkout session, can specify these
 * 3DS values that will override their profile settings
 */
@property (nonatomic, strong) VisaThreeDSSetUp *threeDSSetUp;
@end
```

Merchant information properties are defined as follows:

```
typedef enum VisaUserReviewAction {
    kVisaUserReviewActionContinue = 1,
    kVisaUserReviewActionPay
} VisaUserReviewAction;

@interface VisaMerchantInfo : NSObject
/**
 * Merchant name (preferably merchant application name)
 */
@property (nonatomic, strong) NSString *displayName;

/**
 * Name of merchant's logo image in the bundle
 */
@property (nonatomic, strong) NSString *merchantLogoAssetName;

/**
 * Merchant api key
 */
@property (nonatomic, strong) NSString *merchantAPIKey;
/**
```

```

        Custom review message (truncated if more than 70 characters)
        */
@property (nonatomic, strong) NSString *customReviewActionContinueMessage;

/**
    Set the profile id if needed
    */
@property (nonatomic, strong) NSString *externalProfileId;

/**
    * Flag to indicate passed in amount is the final amount that will be
    * charged to the user
    * kVisaUserReviewActionPay - final
    * kVisaUserReviewActionContinue - subjected to change
    */
@property (nonatomic) VisaUserReviewAction userReviewAction;

/**
    * Set the canadianDebitCardSupport by Merchant if needed. Should be one
    * of the constants defined below -
    * VAcceptCanadianDebitCard or VDenyCanadianDebitCard
    */
@property (nonatomic, assign) NSString *canadianDebitCardSupport;

/**
    * Set the client id if needed
    */
@property (nonatomic, strong) NSString *externalClientId;

/**
    * Add any custom key value pairs if needed
    */
@property (nonatomic, strong) NSDictionary *customData;

/**
    * List of Locale Supported format is Language-CountryCode
    */

@property (nonatomic, strong) NSString *preferredLocale;

/**
    * Tag to provide information if child directed treatment should be
    * applied for Google ad tracking feature
    */
@property (nonatomic) BOOL applyChildDirectedTreatment;

/**
    * Property to disable card scan feature, which enables users to scan card
    * details with their camera
    */
@property (nonatomic) BOOL disableCardScanning;

/**
    * Flag to communicate to user that card data will be added to merchant's
    * records tied with the user
    */
@property (nonatomic) BOOL communicateCardStoredOnFile;

```

```
extern NSString *const VLocale_enAE;
extern NSString *const VLocale_enAU;
extern NSString *const VLocale_enCA;
extern NSString *const VLocale_enCN;
extern NSString *const VLocale_enGB;
extern NSString *const VLocale_enHK;
extern NSString *const VLocale_enIE;
extern NSString *const VLocale_enMY;
extern NSString *const VLocale_enNZ;
extern NSString *const VLocale_enSG;
extern NSString *const VLocale_enUS;
extern NSString *const VLocale_enZA;
extern NSString *const VLocale_esAR;
extern NSString *const VLocale_esCL;
extern NSString *const VLocale_esCO;
extern NSString *const VLocale_esES;
extern NSString *const VLocale_esMX;
extern NSString *const VLocale_esPE;
extern NSString *const VLocale_frCA;
extern NSString *const VLocale_frFR;
extern NSString *const VLocale_plPL;
extern NSString *const VLocale_ptBR;
extern NSString *const VLocale_zhCN;
extern NSString *const VLocale_zhHK;
extern NSString *const VAcceptCanadianDebitCard;
extern NSString *const VDenyCanadianDebitCard;
extern NSString *const VEnableTokenization;
extern NSString *const VIgnoreTokenization;
extern NSString *const VDisableTokenization;

@end
```

Note: If you specify an asset name for the `merchantLogoAssetName` property, you must ensure the asset is added to the project.

The logo size must be 29x112 pixels for non-retina and 58x224 pixels for retina screens.

Syntax

```
– (VisaPaymentInfo*) paymentInfo { ... }
```

Input Parameters

None

Output Parameters

None

Return Value

A pointer to the `VisaPaymentInfo` object.

Example

```
– (VisaPaymentInfo *)paymentInfo {
```



```

VisaPaymentInfo *paymentInfo = [[VisaPaymentInfo alloc] init];

// Uncomment the below lines to pass amount
[paymentInfo setAmount:[NSDecimalNumber
    decimalNumberWithString:@"9.99"]];
[paymentInfo setCurrency:VCurrencyCodeUS];
paymentInfo.merchantRequestId = @"a001";
paymentInfo.subtotal = [NSDecimalNumber decimalNumberWithString:@"0"];
paymentInfo.shippingHandlingCharges =
    [NSDecimalNumber decimalNumberWithString:@"0"];
paymentInfo.tax = [NSDecimalNumber decimalNumberWithString:@"0"];
paymentInfo.discount = [NSDecimalNumber decimalNumberWithString:@"0"];
paymentInfo.giftWrapCharges = [NSDecimalNumber decimalNumberWithString:@"0"];
paymentInfo.miscCharges = [NSDecimalNumber decimalNumberWithString:@"0"];
paymentInfo.paymentDescription = @"sample transaction";
paymentInfo.orderId = @"111";
paymentInfo.promoCode = @"freeCheckout";
paymentInfo.isShippingAddressRequired = YES;
// SUMMARY, FULL (includes account number), or NONE (only call ID)
paymentInfo.datalevel = @"SUMMARY";

// Use an existing call ID from an earlier session
// to directly launch the sign-in flow with the card and addresses
// set in the referenced call ID.
paymentInfo.setReferenceCallId = "...";
// Pass merchant information
VisaMerchantInfo *merchantInfo = [[VisaMerchantInfo alloc] init];
// Uncomment the next line after filling in the API Key
// merchantInfo.merchantAPIKey = [replace with API key];

merchantInfo.userReviewAction = kVisaUserReviewActionContinue;
merchantInfo.displayName = @"My Merchant Name";
merchantInfo.externalProfileId = @"default";
merchantInfo.displayName = @"The Merchant App";
merchantInfo.customReviewActionContinueMessage = @"Add custom message here";
// Uncomment the following line after ensuring that the (mercLogo) asset is added
// to the project:
// merchantInfo.merchantLogoAssetName = @"mercLogo";

merchantInfo.canadianDebitCardSupport = VAcceptCanadianDebitCard;

// Choose a preferred locale when SDK is launched
merchantInfo.preferredLocale = VLocale_enUS;

paymentInfo.merchantInfo = merchantInfo;

// Set up Verified by Visa (3D-Secure)
VisaThreeDSSetUp *threeDSSetUp = [VisaThreeDSSetUp new];
threeDSSetUp.threeDSActive = VisaActivate3DS;
paymentInfo.threeDSSetUp = threeDSSetUp;

// If you do not specify a card brand, shipping region, or billing region
// either here or in your profile, all supported brands and regions will be enabled.

NSMutableArray *cardBrandArray =
    [[NSMutableArray alloc] initWithObjects:@"Visa", @"MasterCard", @"AMEX",

```

```
@("Discover", nil];
paymentInfo.acceptedPaymentTypesAndBrands = cardBrandArray;

// Set accepted countries for shipping

NSMutableArray *acceptedCountryList =
    [[NSMutableArray alloc] initWithObjects:@"US",@"CA", @"AU", nil];
paymentInfo.acceptedShippingCountries = acceptedCountryList;

// Set accepted countries for billing
NSMutableArray *acceptedBillingCountryList =
    [[NSMutableArray alloc] initWithObjects:@"CA", @"US", @"AU" , nil];
paymentInfo.acceptedBillingCountries = acceptedBillingCountryList;

// Specify an external client ID if needed
// merchantInfo.externalClientId = @"MyClientID";

// Set custom data (optional)
paymentInfo.merchantInfo.customData = @{
    @"nvPair" : @[
        @{
            @"name" : @"customName1",
            @"value" : @"customValue1"
        },
        @{
            @"name" : @"customName2",
            @"value" : @"customValue2"
        }
    ]};

// Enable tokenization
paymentInfo.enableTokenization = VEnableTokenization;

...
return paymentInfo;
}
```

VisaServer

Visa Checkout calls the `VisaServer` method to configure the targeted transaction environment at checkout time.

Syntax

```
- (VisaServer)VisaServer { ... }
```

Input Parameters

None

Output Parameters

None

Return Value

One of the following values:

- `kVisaSandboxServer` specifies the Sandbox server.
- `kVisaProductionServer` specifies a production server.

Example

```
- (VisaServer)visaServer {  
    return kVisaSandboxServer;  
}
```

GAISharedInstance

Use the `GAISharedInstance` method to pass the shared GAI instance if Google analytics is implemented in your app

Note: *Contact your Visa Checkout representative for information regarding its use with the Swift framework; name spacing is not applicable.*

Syntax

```
- (id)GAISharedInstance { ... }
```

Input Parameters

None

Output Parameters

None

Return Value

ID of your GAI shared instance

Example

```
- (GAI *)GAISharedInstance {  
    return [GAI sharedInstance];  
}
```

Update Payment Info Pixel Image 5

Update Payment Info Pixel Image Summary

You can confirm a purchase by calling the `payment/updatepaymentinfo.gif`. Specifically, you associate parameters that convey the purchase information from within the app without requiring your shared secret.

You can also use this call to specify card-on-file transactions, in which case the Visa Checkout lightbox is not used.

Note: The `updatepaymentinfo.gif` image itself is 1-pixel.

You must specify the following parameters as part of the image URL:

- Your public API key (`apikey`), which is different than your shared secret or an encrypted key (`encKey`)
- Transaction whose payment you want to confirm (`callId`)

Notes

1. You can calculate shipping, apply discounts, and so on, within the button source itself if you want the consumer to confirm in the lightbox.
2. You must load the image with the query parameters or call `payment/info`.
3. As a best practice, you should load `payment/updatepaymentinfo.gif` when the consumer confirms the order on your confirmation page.
4. You can only specify the order update or payment update in the same operation; you cannot do both in the same operation.
5. You can specify the following event types:

Event description	eventType
Card on file saved (outside of a purchase flow)	Create
Order placed	Confirm
Order placed using a card on file	Confirm_COF
Order rejected by risk/fraud review	Fraud

Order canceled	Cancel
None of the above	Other

Card on File Events

When a consumer uses the lightbox to confirm a purchase, you specify `Confirm` for the `eventType`. For subsequent purchases using the same card, you can use the previously generated call ID and `Confirm_COF` for the `eventType`.

To place a card on file without a purchase, you specify `Create` for the `eventType`; after which, you can specify `Confirm_COF` for the `eventType`.

Promotions

To apply a promotion to a purchase, you must specify values for the following fields:

- Currency code (`currencyCode`)
- Amount of promotion to be deducted from the total (`discount`)
- Event type (`eventType`), which is either `Confirm` or `Confirm_COF`
- Your order ID for the transaction (`orderId`)
- Promotion code, which is provided by Visa Checkout (`promoCode`)
- Subtotal (`subtotal`), which is the amount of the items before applying discounts and such
- Total (`total`), which is the net amount of the transaction after applying discounts and such

Update Payment Info Pixel Image Request

Path and Endpoints

Resource Path: `payment/updatepaymentinfo.gif`

Complete endpoint:

Sandbox:

`https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/updatepaymentinfo.gif`

Live:

`https://secure.checkout.visa.com/wallet-services-web/payment/updatepaymentinfo.gif`

Update Payment Info Pixel Image Request Parameters

Property	Description
apiKey	<p>(Required) Public API key, which is different than the shared secret.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Example: apiKey=xxxxxxxxxxxxxxxxxxxxxxxxxxxx</p> <p>Since 2.0</p>
callId	<p>(Required) Visa Checkout transaction ID returned by the Visa Checkout <code>payment.success</code> event.</p> <p>Format: Alphanumeric; maximum 48 characters</p> <p>Example: "callId": "..."</p> <p>Since 2.0</p>
currencyCode	<p>(Required) The currency with which to process the transaction. Required because <code>total</code> must be provided.</p> <p>Format: It is one of the following ISO 4217 standard alpha-3 code values:</p> <ul style="list-style-type: none"> • ARS - Argentine Peso (Since 2.7) • AUD - Australian Dollar • BRL - Brazilian Real (Since 2.7) • CAD - Canadian Dollar • CNY - Yuan Renminbi (Since 2.7) • CLP - Chilean Peso (Since 2.7) • COP - Colombian Peso (Since 2.7) • EUR - Euro (Since 4.3) • HKD - Hong Kong Dollar (Since 2.7) • MYR - Malaysian Ringgit (Since 2.7) • MXN - Mexican Peso (Since 2.7) • NZD - New Zealand Dollar (Since 2.7) • PEN - Nuevo Sol - Peru (Since 2.7) • PLN - Polish Zloty (Since 4.3) • SGD - Singapore Dollar (Since 2.7) • ZAR - Rand (Since 2.7) • AED - UAE Dirham (Since 2.7) • GBP - UK Pound Sterling (Since 4.3) • USD - US Dollar <p>Currency codes must be uppercase.</p> <p>Example: "currencyCode": "USD"</p> <p>Since 2.0</p>

Property	Description
discount	<p>(Optional) Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "discount" : "2.50"</p> <p>Since 2.0</p>
eventType	<p>(Required) Kind of event associated with the update.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Create - Card on file saved (outside of a purchase flow) • Confirm - Order placed • Confirm_COF - Order placed using a card on file • Cancel - Order canceled • Fraud - Order rejected by risk/fraud review • Other - None of the above <p>Example: "eventType": "Confirm"</p> <p>Since 2.0</p>
giftWrap	<p>(Optional) Total gift-wrapping charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "giftWrap" : "1.99"</p> <p>Since 2.0</p>
misc	<p>(Optional) Total uncategorized charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "misc": "1.00"</p> <p>Since 2.0</p>
orderId	<p>(Optional) Merchant's order ID associated with the payment.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Since 2.0</p>
promoCode	<p>(Optional) Promotion codes associated with the payment.</p> <p>Format: Alphabetic characters, digits, space (), underscore (_), hyphen (-), exclamation point (!), "at" sign (@), pound sign or hash mark (#), dollar sign (\$), percent sign (%), asterisk (*), left/open parenthesis ((), right/close parenthesis ()), and plus sign (+). Multiple promotion codes are separated by a period (.); maximum 100 characters for the entire string</p> <p>Example: "promoCode": "17.15"</p> <p>Since 2.0</p>

Property	Description
reason	<i>(Optional)</i> Reason for the update. Format: Alphanumeric; maximum 255 characters Since 2.0
shippingHandling	<i>(Optional)</i> Total of shipping and handling charges for the payment. Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits Example: "shippingHandling" : "3.00" Since 2.0
subtotal	<i>(Required)</i> Subtotal of the payment. Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits Example: "subtotal": "9.00" Since 2.0
tax	<i>(Optional)</i> Total tax-related charges in the payment. Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits Example: "tax": "1.00" Since 2.0
total	<i>(Required)</i> Total of the payment including all amounts. Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits Example: "total": "9.00" Since 2.0

Update Payment Info Pixel Image Response

The 1-pixel image

Update Payment Data Info Pixel Image Error Messages

An error response contains the `v-message` header that you can use to determine the error. Typically, you will use debugging tools built into the browser to view this message.

Header	Description
v-message	<p>Visa Checkout error message.</p> <p>Format: Alphanumeric</p> <p>Example: "v-message":"callid ff6e689c-170c-4f18-a1ba-da5047a35f152 was not found"</p> <p>Since 2.0</p>

Update Payment Info Pixel Image Examples

Update Payment Info Request

GET

<https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/updatepaymentinfo.gif>
 ?apikey=...
 &callId=...
 ¤cyCode=USD
 &eventType=...
 &discount=...
 &giftWrap=...
 &misc=...
 &orderId=...
 &promoCode=...
 &reason=...
 &subtotal=...
 &shippingHandling=...
 &tax=...
 &total=...

Objective C for iOS

```

- (void) sendUpdatePaymentInfoPixelRequest: (VisaPaymentSummary*) paymentSummary
{
    NSString *baseUrl
        = @"https://sandbox.secure.checkout.visa.com/
           wallet-services-web/payment/updatepaymentinfo.gif";
    NSString *callID = paymentSummary.callId;
    NSString *queryString = [NSString stringWithFormat:
        @"apikey=%@"
        "&callId=%@"
        "&currencyCode=USD"
        "&discount="
        "&eventType=CONFIRM"
        "&giftWrap="
        "&misc="
        "&orderId="
        "&promoCode="
        "&reason="
        "&shippingHandling="
        "&subtotal=10.00"
        "&tax="
        "&total=10.00", merchantAPIKey, callID];

    NSString *finalRequestURL
        = [NSString stringWithFormat:@"%s%@", baseUrl, queryString];
    NSURL *url = [NSURL URLWithString:finalRequestURL];
    NSMutableURLRequest *request
        = [[NSMutableURLRequest alloc] initWithURL:url];
    [request setHTTPMethod:@"GET"];
    [request setTimeoutInterval:60];
    NSOperationQueue *queue = [[NSOperationQueue alloc] init];
    [NSURLConnection sendAsynchronousRequest:request
     queue:queue
     completionHandler:
         ^(NSURLResponse * _Nullable response,
           NSData * _Nullable data,
           NSError * _Nullable connectionError) {
         //Process successful or error conditions
     }]];
}

```

Update Payment Info Error Response

```

v-message:"callid ... was not found

```

Update Payment Info

6

Update Payment Info Summary

Call the `payment/info/` resource path at the appropriate endpoint to confirm the amounts the consumer specified in the Visa Checkout lightbox. You can modify the amounts before calling `payment/info/`. For example, you can up-sell the customer, calculate shipping, apply a discount, and so on, on your confirmation page and change the amounts in your Update Payment Info request.

You can also use this call to specify card-on-file transactions, in which case the Visa Checkout lightbox is not used.

Event Types

Additionally, you can update order information. You can register multiple order and payment events in the same Update Payment Info request; for example, you can include order creation information, payment information, and order confirmation information in the same request. When registering multiple order and payment events, you must specify at least one order event.

The following table shows the typical usage of the order event's type (`eventType`):

Event description	eventType
Card on file saved (outside of a purchase flow)	Create
Order placed	Confirm
Order placed using a card on file	Confirm_COF
Order rejected by risk/fraud review	Fraud
Order canceled	Cancel
None of the above	Other

The following table shows the typical usage of the payment event's type (`eventType`) and status (`eventStatus`):

Event description	eventType	eventStatus
Authorization was successful	Authorize	Success

Authorization was declined	Authorize	Failure
Settlement	Capture	Success
Settlement failure	Capture	Failure
Refund	Refund	Success
Refund failure	Refund	Failure
Chargeback	Chargeback	Chargeback
Refund due to chargeback	Refund	Chargeback
Other	Other	Other

Card on File Events

When a consumer uses the lightbox to confirm a purchase, you specify `Confirm` for the `eventType`. For subsequent purchases using the same card, you can use the previously generated call ID and `Confirm_COF` for the `eventType`.

To place a card on file without a purchase, you specify `Create` for the `eventType`; after which, you can specify `Confirm_COF` for the `eventType`.

Promotions

To apply a promotion to a purchase, you must specify values for the following fields:

- Currency code (`currencyCode`)
- Amount of promotion to be deducted from the total (`discount`)
- Event type (`eventType`), which is either `Confirm` or `Confirm_COF`
- Your order ID for the transaction (`orderId`)
- Promotion code, which is provided by Visa Checkout (`promoCode`)
- Subtotal (`subtotal`), which is the amount of the items before applying discounts and such
- Total (`total`), which is the net amount of the transaction after applying discounts and such

Notes

1. To add a card on file without the consumer making a purchase, specify `Create` as the order event type; in which case, `subtotal` and `total` can be 0.
2. To confirm the total amount of a purchase, specify `Confirm` as the order event type.

Update Payment Info Request

Path and Endpoints

Resource Path: `payment/info/{callId}`

Complete endpoint:

Sandbox:

`https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/{callId}`

Live:

`https://secure.checkout.visa.com/wallet-services-web/payment/info/{callId}`

Parameter	Description
{callId}	The call ID that identifies the transaction. Format: Alphanumeric Since 2.0

Method

PUT

Required Headers

Header	Description
x-pay-token	<p>A token identifying the transaction and its contents.</p> <p>Format: Alphanumeric; maximum 100 characters in the form of x-pay-token: x:UNIX_UTC_Timestamp:SHA256_hash, where</p> <ul style="list-style-type: none"> UNIX_UTC_Timestamp is a UNIX Epoch timestamp SHA256_hash is an SHA256 hash of the following unseparated items: <ol style="list-style-type: none"> Shared secret associated with the API key Timestamp from the transaction; exactly the same as UNIX_UTC_Timestamp Resource path (API name) This HTTPS request's query string, if it exists <p>Note: The query string includes one or more parameters in name-value pair format, in which the name is separated from the value by an equal sign (=). Parameters are separated from each other by an ampersand (&) character. The initial question mark (?) is not included in the query string. The query string must be URL encoded, excepting the following characters, per RFC 3986: hyphen, period, underscore, and tilde. Multiple query parameters must be in alphabetical order.</p> <ol style="list-style-type: none"> Complete request body, when a request body exists <p>Example: x-pay-token: x:1358092911:....</p> <p>Since 2.0</p>
Accept	<p>Acceptable response format.</p> <p>Format: Must include application/json</p> <p>Example: Accept: application/json</p> <p>Since 2.0</p>
Content-Type	<p>Format of the content.</p> <p>Format: Must include application/json</p> <p>Example: Content-Type: application/json</p> <p>Since 2.0</p>

Query Parameters

Parameter	Description
apikey	<p>(Required) Public API key, which is different than the shared secret.</p> <p>Format: Alphanumeric; maximum 49 characters</p> <p>Example: apikey=...</p> <p>Since 2.0</p>

Update Payment Info Request Body

Multiple Info Properties

You can specify multiple order info and payment info properties by specifying multiple property structures in a single `updateInfo` structure.

Note: *An `updateInfo` structure is not required if you update only a single `orderInfo` or `payInfo` structure.*

Property	Description
<code>orderInfo</code>	<p>(Optional) Order info properties, one per <code>orderInfo</code> structure.</p> <p>Format: One or more <code>orderInfo</code> structures.</p> <p>Since 3.2</p>
<code>payInfo</code>	<p>(Optional) Pay info properties structure.</p> <p>Format: A <code>payInfo</code> structure.</p> <p>Since 3.2</p>

Order Info Properties

You specify order info properties in an `orderInfo` structure.

Property	Description
total	<p><i>(Required)</i> Total of the payment including all amounts.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "total": "9.00"</p> <p>Since 2.0</p>
currencyCode	<p><i>(Required)</i> The currency with which to process the transaction. Required because total must be provided.</p> <p>Format: It is one of the following ISO 4217 standard alpha-3 code values:</p> <ul style="list-style-type: none"> • ARS - Argentine Peso (Since 2.7) • AUD - Australian Dollar • BRL - Brazilian Real (Since 2.7) • CAD - Canadian Dollar • CNY - Yuan Renminbi (Since 2.7) • CLP - Chilean Peso (Since 2.7) • COP - Colombian Peso (Since 2.7) • EUR - Euro (Since 4.3) • HKD - Hong Kong Dollar (Since 2.7) • MYR - Malaysian Ringgit (Since 2.7) • MXN - Mexican Peso (Since 2.7) • NZD - New Zealand Dollar (Since 2.7) • PEN - Nuevo Sol - Peru (Since 2.7) • PLN - Polish Zloty (Since 4.3) • SGD - Singapore Dollar (Since 2.7) • ZAR - Rand (Since 2.7) • AED - UAE Dirham (Since 2.7) • GBP - UK Pound Sterling (Since 4.3) • USD - US Dollar <p>Currency codes must be uppercase.</p> <p>Example: "currencyCode": "USD"</p> <p>Since 2.0</p>
subtotal	<p><i>(Required)</i> Subtotal of the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "subtotal": "9.00"</p> <p>Since 2.0</p>

Property	Description
shippingHandling	<p>(Optional) Total of shipping and handling charges for the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "shippingHandling" : "3.00"</p> <p>Since 2.0</p>
tax	<p>(Optional) Total tax-related charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "tax": "1.00"</p> <p>Since 2.0</p>
discount	<p>(Optional) Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "discount" : "2.50"</p> <p>Since 2.0</p>
giftWrap	<p>(Optional) Total gift-wrapping charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "giftWrap" : "1.99"</p> <p>Since 2.0</p>
misc	<p>(Optional) Total uncategorized charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "misc": "1.00"</p> <p>Since 2.0</p>
eventType	<p>(Required) Kind of event associated with the update.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Create - Card on file saved (outside of a purchase flow) • Confirm - Order placed • Confirm_COF - Order placed using a card on file • Cancel - Order canceled • Fraud - Order rejected by risk/fraud review • Other - None of the above <p>Example: "eventType": "Confirm"</p> <p>Since 2.0</p>

Property	Description
orderId	<p><i>(Optional)</i> Merchant's order ID associated with the payment.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Since 2.0</p>
promoCode	<p><i>(Optional)</i> Promotion codes associated with the payment.</p> <p>Format: Alphabetic characters, digits, space (), underscore (_), hyphen (-), exclamation point (!), "at" sign (@), pound sign or hash mark (#), dollar sign (\$), percent sign (%), asterisk (*), left/open parenthesis ((), right/close parenthesis ()), and plus sign (+). Multiple promotion codes are separated by a period (.); maximum 100 characters for the entire string</p> <p>Example: "promoCode": "17.15"</p> <p>Since 2.0</p>
reason	<p><i>(Optional)</i> Reason for the update.</p> <p>Format: Alphanumeric; maximum 255 characters</p> <p>Since 2.0</p>

Pay Info Properties

You specify pay info properties in an `payInfo` structure.

Property	Description
payTransId	<p><i>(Optional)</i> Payment transaction ID associated with the merchant authorization of the payment instrument. Use the identifier for a transaction within the card network. The processor receives this number from the network and passes it on to the merchant in the authorization, capture, refund, etc.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Since 2.0</p>
eventType	<p><i>(Required)</i> Kind of event associated with the update.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Authorize • Capture • Refund • Cancel • Fraud • Chargeback • Other <p>Example: "eventType": "Authorize"</p> <p>Since 2.0</p>

Property	Description
eventStatus	<p>(Required) Status of the event.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Success • Failure • Fraud • Chargeback • Other <p>Specify Success or Failure if the eventType is Authorize, Capture, Refund, or Cancel; otherwise, specify the same value as the eventType, e.g. specify Other if the eventType is Other.</p> <p>Example: "eventStatus": "Success"</p> <p>Since 2.0</p>
currencyCode	<p>(Required) The currency with which to process the transaction. Required because total must be provided.</p> <p>Format: It is one of the following ISO 4217 standard alpha-3 code values:</p> <ul style="list-style-type: none"> • ARS - Argentine Peso (Since 2.7) • AUD - Australian Dollar • BRL - Brazilian Real (Since 2.7) • CAD - Canadian Dollar • CNY - Yuan Renminbi (Since 2.7) • CLP - Chilean Peso (Since 2.7) • COP - Colombian Peso (Since 2.7) • EUR - Euro (Since 4.3) • HKD - Hong Kong Dollar (Since 2.7) • MYR - Malaysian Ringgit (Since 2.7) • MXN - Mexican Peso (Since 2.7) • NZD - New Zealand Dollar (Since 2.7) • PEN - Nuevo Sol - Peru (Since 2.7) • PLN - Polish Zloty (Since 4.3) • SGD - Singapore Dollar (Since 2.7) • ZAR - Rand (Since 2.7) • AED - UAE Dirham (Since 2.7) • GBP - UK Pound Sterling (Since 4.3) • USD - US Dollar <p>Currency codes must be uppercase.</p> <p>Example: "currencyCode": "USD"</p> <p>Since 2.0</p>

Property	Description
total	<i>(Required)</i> Total of the payment including all amounts. Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits Example: "total": "9.00" Since 2.0
authCode	<i>(Optional)</i> Authorization code associated with the transaction. Format: Numeric; maximum 6 digits Example: "authCode": "123456" Since 2.0
avsResponseCode	<i>(Optional)</i> Address verification system response code. Format: Alphanumeric Example: "avsResponseCode" : "V" Since 2.0
reason	<i>(Optional)</i> Reason for the update. Format: Alphanumeric; maximum 255 characters Since 2.0

Update Payment Info Errors

Status	Code	Description
400	1015	Invalid request data; a required field is either missing or invalid
401	1017	The API key used in the operation is not authorized for the requested action; ensure that the API key corresponds to the call ID
403	None	x-pay-token header missing or invalid, or API key is missing or invalid
404	1010	API key or call ID not found, or data referenced by the API key is invalid or not available

Note: Other errors are internal.

Update Payment Info Examples

Order Update Success Example

JSON Request Including Headers

```
PUT
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/03e3b03f-233e-..
apikey=...
Accept: application/json
X-PAY-TOKEN: X:1397847508:...
```

JSON Request Body

```
{
  "orderInfo" : {
    "total" : "101",
    "currencyCode" : "USD",
    "subtotal" : "80.1",
    "shippingHandling" : "5.1",
    "tax" : "7.1",
    "discount" : "5.25",
    "giftWrap" : "10.1",
    "misc" : "3.2",
    "eventType" : "Confirm",
    "orderId" : "testorderId",
    "promoCode" : "testPromoCode",
    "reason" : "Order Successfully Created"
  }
}
```

JSON Response

```
HTTP/1.1 200 OK
```

```
{
}
```

XML Request Including Headers

```
PUT
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/03e3b03f-233e-..
apikey=...
Accept: application/xml
X-PAY-TOKEN: X:1397847508:...
```

XML Request Body

```
<?xml version="1.0" encoding="UTF-8"?>
<p:updatePaymentInfoRequest
xmlns:p="http://www.visa.com/vme/walletservices/external/payment"
xmlns:p1="http://www.visa.com/vme/walletservices/external/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.visa.com/vme/walletservices/external/payment
                    payment_mgmt.xsd ">
<p:orderInfo>
<p1:currencyCode>USD</p1:currencyCode>
<p1:subtotal>80</p1:subtotal>
<p1:shippingHandling>5</p1:shippingHandling>
<p1:tax>7</p1:tax>
<p1:discount>5</p1:discount>
<p1:giftWrap>10</p1:giftWrap>
<p1:misc>3</p1:misc>
<p1:total>100</p1:total>
<p:eventType>Confirm</p:eventType>
<p1:orderId>testorderId</p1:orderId>
<p1:promoCode>testCampaignId</p1:promoCode>
<p:reason>Received money</p:reason>
</p:orderInfo>
</p:updatePaymentInfoRequest>
```

XML Response

HTTP/1.1 200 OK

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:updatePaymentInfoResponse
...
/>
```

Payment Update Example**JSON Request Including Headers**

```
PUT
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/03e3b03f-...?
apikey=...
Accept: application/json
X-PAY-TOKEN: X:1397847508:...
```

JSON Request Body

```
{
  "payInfo" : {
    "payTransId" : "66012345001069003",
    "eventType" : "Authorize",
    "eventStatus" : "Success",
    "currencyCode" : "USD",
    "total" : "100",
    "authCode" : "123456",
    "avsResponseCode" : "V",
    "reason" : "..."
  }
}
```

JSON Response

HTTP/1.1 200 OK

```
{
}
```

XML Request Including Headers

```
PUT
https://sandbox.secure.checkout.visa.com/wallet-services-web/payment/info/03e3b03f-233e-..
apikey=...
Accept: application/xml
X-PAY-TOKEN: X:1397847508:...
```

XML Request Body

```
<p:updatePaymentInfoRequest
xmlns:p="http://www.visa.com/vme/walletservices/external/payment"
xmlns:pl="http://www.visa.com/vme/walletservices/external/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.visa.com/vme/walletservices/external/payment
payment_mgmt.xsd ">
  <p:payInfo>
    <p:payTransId>12332432</p:payTransId>
    <p:eventType>Authorize</p:eventType>
    <p:eventStatus>Success</p:eventStatus>
    <p:currencyCode>USD</p:currencyCode>
    <p:total>888.32</p:total>
    <p:authCode>123456</p:authCode>
    <p:avsResponseCode>Y</p:avsResponseCode>
    <p:reason>Authorization was successful</p:reason>
  </p:payInfo>
</p:updatePaymentInfoRequest>
```

XML Response

HTTP/1.1 200 OK

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:updatePaymentInfoResponse
  ...
/>
```

Update Payment Info Error Examples

```
{ "responseStatus" :
{ "status" : 404,
  "code" : "1010",
  "severity" : "ERROR",
  "message" : "CallId b9346ed5-08d1-44b2-be32-bbde5c4bf34f was not found."
} }
```

```
{
"responseStatus":
{
  "status":400,
  "code":1015,
  "severity":"INFO",
  "message":"Invalid request data. "
}
}
```

Update Payment Info Error Example

```
<responseStatus>
<status>400</status>
<code>1015</code>
<severity>INFO</severity>
<message>Invalid request data. </message></responseStatus>
</errorResponse>
```


Update Multiple Info Structure Examples

JSON Request Body

```
{
  "updateInfo": [
    {
      "orderInfo": {
        "currencyCode": "USD",
        "discount": 1.11,
        "eventType": "Create",
        "giftWrap": 20,
        "misc": 333.00,
        "orderId": 1234556666,
        "promoCode": "c3444ffttt",
        "reason": "Because order got created",
        "shippingHandling": 51.99,
        "subtotal": 800,
        "tax": 71,
        "total": 1000.11
      }
    },
    {
      "payInfo": {
        "reason": "Just a test",
        "avsResponseCode": "Y",
        "total": 88.80,
        "currencyCode": "USD",
        "eventStatus": "Success",
        "eventType": "Authorize",
        "payTransId": "123123"
      }
    },
    {
      "orderInfo": {
        "currencyCode": "USD",
        "discount": 1.11,
        "eventType": "Confirm",
        "giftWrap": 20,
        "misc": 333.00,
        "orderId": 1234556666,
        "promoCode": "c3444ffttt",
        "reason": "Because order got approved",
        "shippingHandling": 51.99,
        "subtotal": 800,
        "tax": 71,
        "total": 1000.11
      }
    }
  ]
}
```

JSON Response

HTTP/1.1 200 OK

{
}**XML Request**

```

<?xml version="1.0" encoding="UTF-8"?>
<p:updatePaymentInfoRequest
  xmlns:p="http://www.visa.com/vme/walletservices/external/payment"
  xmlns:pl="http://www.visa.com/vme/walletservices/external/common"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.visa.com/vme/walletservices/external/payment payment_mgmt.xsd">
  <p:updateInfo>
    <p:orderInfo>
      <p1:currencyCode>USD</p1:currencyCode>
      <p1:discount>1.11</p1:discount>
      <p:eventType>Create</p:eventType>
      <p1:giftWrap>20</p1:giftWrap>
      <p1:misc>333.12</p1:misc>
      <p1:orderId>1234556666</p1:orderId>
      <p1:promoCode>c3444ffttt</p1:promoCode>
      <p:reason>Order got created</p:reason>
      <p1:shippingHandling>51.99</p1:shippingHandling>
      <p1:subtotal>800</p1:subtotal>
      <p1:tax>71</p1:tax>
      <p1:total>1000.11</p1:total>
    </p:orderInfo>
  </p:updateInfo>
  <p:updateInfo>
    <p:payInfo>
      <p:payTransId>11100001122222222</p:payTransId>
      <p:eventType>Authorize</p:eventType>
      <p:eventStatus>Success</p:eventStatus>
      <p:currencyCode>USD</p:currencyCode>
      <p:total>200.32</p:total>
      <p:reason>Order got approved</p:reason>
    </p:payInfo>
  </p:updateInfo>
</p:updatePaymentInfoRequest>

```

XML Response

```

HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8"?>
<ns2:updatePaymentInfoResponse
  ...
/>

```

Consumer Information

7

Consumer information is returned in JSON format. You must not rely on the position of structures or fields within the payload as being fixed, nor should you rely upon the existence of fields, such as fields are contextually inapplicable, because they may not be returned. You should consider using standard libraries to parse JSON objects.

The following consumer information is available, either encrypted in a payload or as summary information from Get Payment Data:

Property	Description
paymentRequest	Payment request information from the Visa Checkout library initialization. Format: PaymentRequest Since 2.0
userData	Consumer payment information. Format: UserData Since 2.0
creationTimeStamp	Payment creation timestamp. Format: UNIX Epoch timestamp, in milliseconds. Example: "creationTimeStamp": "1397847423768" Since 2.0
paymentInstrument	Consumer's account information. Contents vary depending on whether enablePANAccess was set to true when the merchant was onboarded and whether the value of dataLevel is FULL or SUMMARY. Note: No consumer information is provided if the dataLevel is NONE. Format: PaymentInstrument Since 2.0

Property	Description
shippingAddress	Shipping address information. Format: Address; see Since 2.0
riskData	Risk information. Format: RiskData Since 2.0
threeDS	Verified by Visa (3-D Secure) information. Format: ThreeDS Since 2.7
visaCheckoutGuest	Guest Checkout. Do not use. Format: It is the following value: <ul style="list-style-type: none"> false Example: "visaCheckoutGuest": "false" Since 3.5
partialShippingAddress	Partial shipping address. Format: PartialShippingAddress Since 2.0

User Data Properties

Property	Description
userFirstName	Consumer's given (first) name. Format: Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), " (double quote), . (period), and – (hyphen); maximum 256 characters Example: "userFirstName" : "Joe" Since 2.0
userLastName	Consumer's surname (last name). Format: Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), " (double quote), . (period), and – (hyphen); maximum 256 characters Example: "userLastName" : "Tester" Since 2.0

Property	Description
userFullName	<p>Concatenation of consumer's first and last names.</p> <p>Format: Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), " (double quote), . (period), and - (hyphen); maximum 256 characters</p> <p>Example: "userFullName" : "Joe Tester"</p> <p>Since 2.9</p>
userName	<p>User name.</p> <p>Format: Alphanumeric, valid email address or mobile phone number; maximum 256 characters</p> <p>Example: "email" : "testuser@mycompany.com"</p> <p>Since 2.0</p>
encUserId	<p>Encoded user ID.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Example: "encUserId" : "..."</p> <p>Since 2.0</p>
userEmail	<p>Valid email address for the consumer making the payment.</p> <p>Format: Alphanumeric, valid email address; maximum 256 characters</p> <p>Example: "email" : "testuser@mycompany.com"</p> <p>Since 2.0</p>
userMobile	<p>Valid mobile phone number for the consumer making the payment; for future use.</p> <p>Format: Numeric or hyphens, parentheses, period, or plus sign, valid for the country; maximum 30 characters</p> <p>Example: "userMobile" : "..."</p> <p>Since 2.0</p>
userPersonalId	<p>Personal ID (Brazil only); it is the CPF (Cadastrado de Pessoas Físicas) tax registration number. It is only available if the merchant is configured by Visa Checkout to receive full information, e.g. onboarded with PAN access enabled.</p> <p>Format: Numeric; maximum 11 digits</p> <p>Example: "userPersonalId": "56453472856"</p> <p>Since 3.3</p>

Payment Request Properties

Property	Description
merchantRequestId	<p>Merchant's ID associated with the request. Visa Checkout stores this value for your use as a convenience.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Since 2.0</p>
currencyCode	<p>The currency with which to process the transaction.</p> <p>Format: It is one of the following ISO 4217 standard alpha-3 code values:</p> <ul style="list-style-type: none"> • ARS - Argentine Peso (Since 2.7) • AUD - Australian Dollar • BRL - Brazilian Real (Since 2.7) • CAD - Canadian Dollar • CNY - Yuan Renminbi (Since 2.7) • CLP - Chilean Peso (Since 2.7) • COP - Colombian Peso (Since 2.7) • EUR - Euro (Since 4.3) • HKD - Hong Kong Dollar (Since 2.7) • MYR - Malaysian Ringgit (Since 2.7) • MXN - Mexican Peso (Since 2.7) • NZD - New Zealand Dollar (Since 2.7) • PEN - Nuevo Sol - Peru (Since 2.7) • PLN - Polish Zloty (Since 4.3) • SGD - Singapore Dollar (Since 2.7) • ZAR - Rand (Since 2.7) • AED - UAE Dirham (Since 2.7) • GBP - UK Pound Sterling (Since 4.3) • USD - US Dollar <p>Currency codes must be uppercase.</p> <p>Example: "currencyCode": "USD"</p> <p>Since 2.0</p>
subtotal	<p>Subtotal of the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "subtotal": "9.00"</p> <p>Since 2.0</p>

Property	Description
shippingHandling	<p>Total of shipping and handling charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "shippingHandling" : "3.00"</p> <p>Since 2.0</p>
tax	<p>Total tax-related charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "tax": "1.00"</p> <p>Since 2.0</p>
discount	<p>Total of discounts related to the payment. If provided, it is a positive value representing the amount to be deducted from the total.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "discount" : 2.50"</p> <p>Since 2.0</p>
giftWrap	<p>Total gift-wrapping charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "giftWrap" : "1.99"</p> <p>Since 2.0</p>
misc	<p>Total uncategorized charges in the payment.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "misc": "1.00"</p> <p>Since 2.0</p>
total	<p>Total of the payment including all amounts.</p> <p>Format: Numeric; maximum 9 digits before an optional decimal point and 4 decimal digits</p> <p>Example: "total": "9.00"</p> <p>Since 2.0</p>
description	<p>The description of the payment. Not yet implemented.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Since 2.0</p>
orderId	<p>Merchant's order ID associated with the payment.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Since 2.0</p>

Property	Description
promoCode	<p>(Optional) Promotion codes associated with the payment.</p> <p>Format: Alphabetic characters, digits, space (), underscore (_), hyphen (-), exclamation point (!), "at" sign (@), pound sign or hash mark (#), dollar sign (\$), percent sign (%), asterisk (*), left/open parenthesis ((), right/close parenthesis ()), and plus sign (+). Multiple promotion codes are separated by a period (.); maximum 100 characters for the entire string</p> <p>Example: "promoCode": "17.15"</p> <p>Since 2.0</p>
customData	<p>Merchant-supplied data, as name-value pairs.</p> <p>Format: Alphanumeric; maximum 1024 characters</p> <p>Since 2.0</p>

Payment Instrument Properties

Field	Description
id	<p>Unique internal ID associated with the payment instrument.</p> <p>Format: Alphanumeric</p> <p>Example: "id" : "..."</p> <p>Since 2.0</p>
lastFourDigits	<p>Last 4 digits of the payment instrument.</p> <p>Format: Numeric; maximum 4 digits</p> <p>Example: "lastFourDigits" : "4448"</p> <p>Since 3.0</p>
tokenInfo	<p>Token information; only available for token-enabled payment instruments.</p> <p>Format: tokenInfo</p> <p>Since 3.4</p>
cryptogramInfo	<p>Cryptogram information associated with the token; only available for tokenized payment instruments.</p> <p>Format: cryptogramInfo</p> <p>Since 3.4</p>
binSixDigits	<p>First 6 digits of account number-based payment instrument.</p> <p>Note: Not present for tokenized payment instruments.</p> <p>Format: Numeric; maximum 6 digits</p> <p>Example: "binSixDigits" : "444444"</p> <p>Since 3.0</p>

Field	Description
accountNumber	<p>Account number associated with the payment instrument, which is available only for account number-based payment instruments. Additionally, it is only available to merchants with permission to request full information, e.g. onboarded with PAN access; accountNumber is not present in summary information.</p> <p>Note: Not present for tokenized payment instruments.</p> <p>Format: Numeric; maximum 19 digits</p> <p>Example: "accountNumber" : "..."</p> <p>Since 2.0</p>
paymentType	<p>Kind of payment instrument.</p> <p>Format: PaymentType</p> <p>Since 2.0</p>
billingAddress	<p>Billing address associated with the payment instrument.</p> <p>Format: Address; see</p> <p>Since 2.0</p>
verificationStatus	<p>Visa Checkout verification status for the payment instrument.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • VERIFIED • NOT_VERIFIED • CONSUMER_OVERRIDE" <p>Example: "verificationStatus" : "VERIFIED"</p> <p>Since 2.0</p>
expired	<p>Whether the card has expired.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • true - Expired • false - Not expired <p>Example: "expired" : "false"</p> <p>Since 2.0</p>
cardArts	<p>Card art information.</p> <p>Format: cardArts</p> <p>Since 2.0</p>
issuerBid	<p>Issuer BID.</p> <p>Format: Alphanumeric; maximum 100 characters</p> <p>Example: "issuerBid" : "14"</p> <p>Since 2.0</p>

Field	Description
nickName	Nick name associated with the payment instrument. Format: Alphanumeric; maximum 140 characters Example: "nickName" : "Fav Visa" Since 2.0
nameOnCard	Name of the consumer on the card. Format: Alphabetic or the following characters: spaces, ' (single quote), ` (back tick), ~ (tilde), " (double quote), . (period), and - (hyphen); maximum 256 characters Example: "nameOnCard" : "John Tester" Since 2.0
cardFirstName	Consumer's first name on card. Format: Alphanumeric; maximum 256 characters Example: "cardFirstName" : "John" Since 2.9
cardLastName	Consumer's last name on card. Format: Alphanumeric; maximum 256 characters Example: "cardLastName" : "Tester" Since 2.9
expirationDate	Payment instrument's expiration date. Note: <i>This expiration date is only provided for account number-based payment instruments. For tokenized payment instruments, the tokenInfo structure contains the token's expiration date.</i> Format: expirationDate Since 2.0
riskData	Risk information related to the transaction, if available. Format: RiskData Since 2.0

Token Info Properties

Token information is only available for token-enabled payment instruments. You must be configured by Visa Checkout to receive this information.

Field	Description
token	<p>The token value associated with the payment instrument. It is only available to merchants with permission to request full information, e.g. onboarded with PAN access; not present in summary information.</p> <p>Format: Numeric; maximum 16 digits</p> <p>Example: "token" : "..."</p> <p>Since 3.4</p>
tokenRange	<p>First 9 digits of the token.</p> <p>Format: Numeric; maximum 9 digits</p> <p>Example: "tokenRange" : "123444444"</p> <p>Since 3.4</p>
last4	<p>Last 4 digits of the token.</p> <p>Format: Numeric; maximum 4 digits</p> <p>Example: "last4" : "1234"</p> <p>Since 3.4</p>
expirationDate	<p>Token's expiration date.</p> <p>Format: expirationDate</p> <p>Since 3.4</p>

Cryptogram Info Properties

Cryptogram information is only available for token-enabled payment instruments. You must be configured by Visa Checkout to receive this information.

Field	Description
cryptogram	<p>Current cryptogram associated with the token, which is Base64 encoded.</p> <p>Note: <i>Visa Checkout creates a new cryptogram with every payload request via Get Payment Data API. To adhere to the VisaNet requirements for Field 126.9, which is 20 bytes long, cryptogram must be decoded into a 20 byte binary value prior to submitting it to VisaNet.</i></p> <p>Format: Alphanumeric</p> <p>Example: "cryptogramInfo" : "..."</p> <p>Since 3.4</p>
eci	<p>An e-commerce indicator (ECI); for example, a successful authentication of a Visa token returns 07 for eci.</p> <p>Format: One of the following values:</p> <ul style="list-style-type: none"> 07 <p>Example: "eci" : "07"</p> <p>Since 3.4</p>

Payment Type Properties

Field	Description
cardBrand	<p>Brand of payment instrument.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none">• VISA• MASTERCARD• AMEX• DISCOVER• ELECTRON (Brazil only; since 3.9)• ELO (Brazil only; since 3.9) <p>Example: "cardBrand" : "VISA"</p> <p>Since 2.0</p>
cardType	<p>Kind of card.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none">• CREDIT• DEBIT• CHARGE• PREPAID• DEFERRED_DEBIT• NONE <p>Example: "cardType" : "CREDIT"</p> <p>Since 2.0</p>

Address

The following information describes address payload fields, irrespective of country. For country-specific formats, see .

Property	Description
id	<p>Address ID</p> <p>Note: Only available for shipping addresses</p> <p>Format: Alphanumeric; maximum 36 characters</p> <p>Example: "id": "..."</p> <p>Since 2.0</p>
verificationStatus	<p>Visa Checkout verification status of the address.</p> <p>Note: Only available for shipping addresses</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • VERIFIED • NOT_VERIFIED • CONSUMER_OVERRIDE <p>Example: "verificationStatus" : "VERIFIED"</p> <p>Since 2.0</p>
personName	<p>Addressee's complete name.</p> <p>Format: Alphanumeric; maximum 256 characters</p> <p>Example: "personName" : "Test User"</p> <p>Since 2.0</p>
firstName	<p>Addressee's first name.</p> <p>Format: Alphanumeric; maximum 256 characters</p> <p>Example: "firstName" : "Test"</p> <p>Since 2.9</p>
lastName	<p>Addressee's last name.</p> <p>Format: Alphanumeric; maximum 256 characters</p> <p>Example: "lastName" : "User"</p> <p>Since 2.9</p>
line1	<p>First line of the address.</p> <p>Format: Alphanumeric; maximum 140 characters</p> <p>Example: "line1" : "1 Main Street"</p> <p>Since 2.0</p>
streetNumber	<p>Street number in the first line of the address, if it exists. It is used only for Brazilian addresses.</p> <p>Format: Alphanumeric; maximum 140 characters</p> <p>Example: "streetNumber": "1738"</p> <p>Since 3.9</p>

Property	Description
streetName	Street name in the first line of the address, if it exists. It is used only for Brazilian addresses. Format: Alphanumeric; maximum 140 characters Example: "streetName": "Haddock Lobo" Since 3.9
additionalLocation	Additional location information in the first line of the address, if it exists. It is used only for Brazilian addresses. Format: Alphanumeric; maximum 140 characters Example: "additionalLocation": "Apt 4" Since 3.9
line2	Second line of the address, if it exists. Format: Alphanumeric; maximum 140 characters Example: "line2" : "..." Since 2.0
line3	Third line of the address, if it exists. It is used only to hold the name of the suburb for New Zealand addresses. Format: Alphanumeric; maximum 140 characters Example: "line3" : "..." Since 2.7
city	City associated with the address. Format: Alphanumeric; maximum 100 characters Example: "city" : "San Francisco" Since 2.0
stateProvinceCode	State or province code associated with the address. Must be a valid 2-character code for US and CA and a valid 3-character code for AU. Format: Alphanumeric; maximum 100 characters Example: "stateProvinceCode" : "CA" Since 2.0

Property	Description
postalCode	<p>Postal code associated with the address, such as a zip code.</p> <p>Format: Depends on stateProvinceCode, maximum 100 characters:</p> <ul style="list-style-type: none"> • US must be 5 digits • CA must be 6 characters separated by a space or a hyphen, e.g. A0A 0A0 • AU must be 4 digits • NZ must be 4 digits <p>Other postal codes must be valid for their respective countries, if a code exists.</p> <p>Example: "postalCode" : "94301"</p> <p>Since 2.0</p>
countryCode	<p>Country code associated with the address.</p> <p>Format: One of the following ISO-3166-1 alpha-2 standard codes:</p> <ul style="list-style-type: none"> • AR - Argentina (Since 2.7) • AU - Australia • BR - Brazil (Since 2.7) • CA - Canada • CN - China (Since 2.9) • CL - Chile (Since 2.9) • CO - Colombia (Since 2.9) • FR - France (Since 4.3) • HK - Hong Kong (Since 2.9) • IE - Ireland (Since 4.3) • MY - Malaysia (Since 2.9) • MX - Mexico (Since 2.9) • PE - Peru (Since 2.9) • PL - Poland (Since 4.3) • NZ - New Zealand (Since 2.9) • SG - Singapore • ZA - South Africa (Since 2.9) • ES - Spain (Since 4.3) • AE - United Arab Emirates (Since 2.9) • GB - United Kingdom (Since 4.3) • US - United States <p>Example: "countryCode" : "US"</p> <p>Since 2.0</p>

Property	Description
phone	Phone number associated with the address. Format: Numeric or hyphens, parentheses, period, or plus sign, valid for the country; maximum 30 characters Example: "phone" : "4155551212" Since 2.0
default	Whether this is the default address. Format: It is one of the following Boolean values: <ul style="list-style-type: none">• true• false Example: "default" : false Since 2.5

Card Arts

Zero or more Card Art structures

Card Art

Property	Description
baseImageFileName	URL to the card art. Format: Alphanumeric, valid URL; maximum 100 characters Important: The URL to the card art is provided for the sole purpose of displaying the card with Visa Checkout and for no other purpose. Example: "baseImageFileName" : "..." Since 2.0
height	Height of art, in pixels. Format: Numeric; value between 1 and 4096, inclusive Example: "height" : ... Since 2.0
width	Width of art, in pixels. Format: Numeric; value between 1 and 4096, inclusive Example: "width" : ... Since 2.0

Expiration Date

Field	Description
month	<p>Expiration month of the payment instrument.</p> <p>Format: The month in MM format, including leading 0 if necessary; from 01 to 12, inclusive.</p> <p>Example: "month" : "09"</p> <p>Since 3.0</p>
year	<p>Expiration year of the payment instrument.</p> <p>Format: The year in YYYY format.</p> <p>Example: "year" : "2015"</p> <p>Since 2.0</p>

Risk Properties

Property	Description
advice	<p>Not currently available. Risk advice for use with your fraud model.</p> <p>Important:</p> <p>The returned value indicates a category of risk and is strictly advisory. Its value should only be used in conjunction with your own experience, models, and risk tolerance to determine whether to complete the transaction.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> LOW - Lower than a medium level of risk anticipated MEDIUM - Medium level of risk anticipated HIGH - Higher than a medium level of risk anticipated UNAVAILABLE - No information available <p>Example: "advice" : "LOW"</p> <p>Since 2.0</p>
score	<p>Risk score; 0 indicates unavailable. Not currently available; always 0. The higher the score, the higher the perceived risk.</p> <p>Important:</p> <p>The returned value indicates a relative value of risk and is strictly advisory. Use this value in conjunction with your own experience, models, and risk tolerance to determine whether to complete the transaction.</p> <p>Format: Numeric; whole value between 0 and 99, inclusive</p> <p>Example: "score" : "0"</p> <p>Since 2.0</p>

Property	Description
avsResponseCode	<p>Address verification system response code.</p> <p>Note: Although AVS is verified when a card is added to the Visa Checkout account, verification information may not always be available in the consumer information payload; in which case, 'unavailable (0)' is returned for the value.</p> <p>Format:Alphanumeric</p> <p>Example: "avsResponseCode" : "V"</p> <p>Since 2.0</p>
cvvResponseCode	<p>Card validation verification system response code.</p> <p>Note: Although a card security code, e.g. CVV2, is verified when a card is added to the Visa Checkout account, verification information may not always be available in the consumer information payload; in which case, 'unavailable (0)' is returned for the value.</p> <p>Format:Alphanumeric</p> <p>Example: "cvvResponseCode" : "M"</p> <p>Since 2.0</p>
ageOfAccount	<p>Number of days since the Visa Checkout account was created, if applicable. You can use it for fraud evaluation; it may not be used for any secondary purpose except as permitted under your Visa Checkout services agreement or with consumer consent.</p> <p>Format:Numeric</p> <p>Example: "ageOfAccount" : 300</p> <p>Since 2.8</p>

3-D Secure Authentication Data Fields

Fields are returned in a `threeDS` structure, which is typically available when the merchant has been configured to use 3-D Secure (Verified by Visa); however, the structure also can be returned on first use of a Visa (Verified by Visa) or Mastercard (SecureCode) card by a European consumer for configured merchants in Europe, regardless of whether 3-D Secure is active. If any field is returned, all fields are returned; however, any field can be empty.

Important:

You must provide some or all of these fields in the authentication message to your processor. Consult with your processor about the fields and values to include in the authentication message.

Field	Description
eciRaw	<p>A brand-specific e-commerce indicator (ECI); for example, a successful authentication of a Visa card returns 05 for eciRaw.</p> <p>Format: One of the following values:</p> <ul style="list-style-type: none"> • 00 - Mastercard (on consumer first use in Europe only) • 01 - Mastercard (on consumer first use in Europe only) • 02 - Mastercard (on consumer first use in Europe only) • 05 - Visa • 06 - Visa • 07 - Visa <p>Example: "eciRaw" : "05"</p> <p>Since 2.7</p>
cavv	<p>Encoded Cardholder Authentication Verification Value or Authentication Verification Value (AVV); returned only for Verified by Visa transactions. This value will be encoded according to the merchant's configuration using either Base64 or Hex encoding. It should be included in the payment authentication request.</p> <p>Format: Alphanumeric; maximum 48 characters</p> <p>Example: "cavv" : "..."</p> <p>Since 2.7</p>
veresEnrolled	<p>Whether the card holder is enrolled in Verified by Visa and the card issuing bank is participating in Verified by Visa. Only a value of Y indicates authentication eligibility.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Y - Enrolled • N - Not enrolled • B - Authentication bypassed • U - Authentication unavailable <p>Example: "veresEnrolled" : "U"</p> <p>Since 2.7</p>
veresTimestamp	<p>VERes response timestamp in Coordinated Universal Time (UTC), also known as Zulu time.</p> <p>Format: ISO 8601 standard in the form of yyyy-mm-ddThh:mm:ss:mmmZ.</p> <p>Example: "veresTimestamp": "2014-12-29T18:34:45.489Z"</p> <p>Since 2.7</p>

Field	Description
paResStatus	<p>Whether the transaction was successfully authenticated.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Y - Authenticated • N - Not authenticated • U - Authentication could not be completed • A - Successful attempts transaction <p>Example: "paResStatus" : "U"</p> <p>Since 2.7</p>
paResTimestamp	<p>PARes response timestamp in Coordinated Universal Time (UTC), also known as Zulu time.</p> <p>Format: ISO 8601 standard in the form of yyyy-mm-ddThh:mm:ss:mmmZ.</p> <p>Example: "paResTimestamp": "2014-12-29T18:37:07.413Z"</p> <p>Since 2.7</p>
signatureVerification	<p>Whether the PARes has been validated successfully.</p> <p>Format: It is one of the following values:</p> <ul style="list-style-type: none"> • Y - Indicates that the signature of the PARes has been validated successfully. • N - Indicates that the PARes could not be validated. <p>Example: "signatureVerification" : "Y"</p> <p>Since 3.5</p>
xId	<p>Gateway or processor's authentication transaction ID, if available. This value will be encoded according to the merchant's configuration using either Base64 or Hex encoding. It should be included in the payment authorization request.</p> <p>Format: Alphanumeric; maximum 40 characters</p> <p>Example: "xId": "..."</p> <p>Since 2.7</p>

Decrypting Consumer Information

A

Consumer information that might be needed to complete a payment is returned by a `payment.success` event. It is always encrypted. The Get Payment Info API returns the same information; however, only full information, which includes the consumer's account number, is encrypted.

Consumer Information Decryption Algorithm

You must decrypt the information using an encrypted dynamic key, which is also returned. To decrypt consumer information:

1. Decrypt the dynamic key:
 - a. Base64-decode the encrypted dynamic key
 - b. Remove the first 32 bytes of the decoded value—this is the HMAC. Calculate a SHA-256 HMAC of the rest of the decoded data using your shared secret and compare it with the HMAC from the first 32 bytes.
 - c. The next 16 bytes should be removed and used as the IV for the decryption algorithm
 - d. Decrypt the remaining data using AES-256-CBC, the IV from Step 1c, and the SHA-256 hash of the shared secret.
2. Decrypt the payment data payload (`encPaymentData`) using the decrypted dynamic key from Step 1:
 - a. Base64-decode the encrypted payment data
 - b. Remove the first 32 bytes of the decoded value—this is the HMAC. Calculate a SHA-256 HMAC of the rest of the decoded data using your decrypted dynamic key and compare it with the HMAC from the first 32 bytes.
 - c. The next 16 bytes should be removed and used as the IV for the decryption algorithm.
 - d. Decrypt the rest of the payload using AES-256-CBC, the IV from Step 2c, and the SHA-256 hash of the decrypted dynamic key.

Note: All plain text data is UTF-8 encoded.

Consumer Information Decryption Examples

Java Decryption Example

The following Java code, using the Bouncy Castle API and `bcprov-jdk15on-149.jar`, provides an example of decrypting the payload.

Note: *Encryption in Java requires Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files.*

```
private static final String CIPHER_ALGORITHM = "AES/CBC/PKCS5Padding";
private static final String HASH_ALGORITHM = "SHA-256";
private static final String HMAC_ALGORITHM = "HmacSHA256";
private static final int IV_LENGTH = 16, HMAC_LENGTH = 32;
private static final Charset utf8 = Charset.forName("UTF-8");
private static final Provider bcProvider;
static {
    bcProvider = new BouncyCastleProvider();
    if (Security.getProvider(BouncyCastleProvider.PROVIDER_NAME) == null) {
        Security.addProvider(bcProvider);
    }
}

private static byte[] decrypt(byte[] key, byte[] data) throws GeneralSecurityException {
    byte[] decodedData = Base64.decode(data);
    if (decodedData == null || decodedData.length <= IV_LENGTH) {
        throw new RuntimeException("Bad input data.");
    }
    byte[] hmac = new byte[HMAC_LENGTH];
    System.arraycopy(decodedData, 0, hmac, 0, HMAC_LENGTH);
    if (!Arrays.equals(hmac,
        hmac(key, decodedData, HMAC_LENGTH, decodedData.length - HMAC_LENGTH))) {
        throw new RuntimeException("HMAC validation failed.");
    }
    byte[] iv = new byte[IV_LENGTH];
    System.arraycopy(decodedData, HMAC_LENGTH, iv, 0, IV_LENGTH);
    Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM, bcProvider);
    cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(hash(key), "AES"),
        new IvParameterSpec(iv));
    return cipher.doFinal(decodedData, HMAC_LENGTH + IV_LENGTH,
        decodedData.length - HMAC_LENGTH - IV_LENGTH);
}

private static byte[] hash(byte[] key) throws NoSuchAlgorithmException {
    MessageDigest md = MessageDigest.getInstance(HASH_ALGORITHM);
    md.update(key);
    return md.digest();
}

private static byte[] hmac(byte[] key, byte[] data, int offset, int length)
    throws GeneralSecurityException {
    Mac mac = Mac.getInstance(HMAC_ALGORITHM, bcProvider);
    mac.init(new SecretKeySpec(key, HMAC_ALGORITHM));
    mac.update(data, offset, length);
    return mac.doFinal();
}
```

C# Decryption Example

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
class Decrypt {
    const int HMAC_LENGTH = 32, IV_LENGTH = 16;
    public static String decryptPayload(String key, String wrappedKey, String payload) {
        return Encoding.UTF8.GetString(decrypt(decrypt(Encoding.UTF8.GetBytes(key),
            Convert.FromBase64String(wrappedKey)), Convert.FromBase64String(payload)));
    }
    public static byte[] decrypt(byte[] key, byte[] data) {
        if (data == null || data.Length <= IV_LENGTH + HMAC_LENGTH) {
            throw new ArgumentException("Bad input data", "data");
        }
        byte[] hmac = new byte[HMAC_LENGTH];
        Array.Copy(data, 0, hmac, 0, HMAC_LENGTH);
        byte[] iv = new byte[IV_LENGTH];
        Array.Copy(data, HMAC_LENGTH, iv, 0, IV_LENGTH);
        byte[] payload = new byte[data.Length - HMAC_LENGTH - IV_LENGTH];
        Array.Copy(data, HMAC_LENGTH + IV_LENGTH, payload, 0, payload.Length);
        //if (byteArrayEquals(hmac, dohmac(key, byteArrayConcat(iv, payload)))) {
        // TODO: Handle HMAC validation failure
        //}
        Aes aes = new AesManaged();
        aes.BlockSize = 128;
        aes.KeySize = 256;
        aes.Key = hash(key);
        aes.IV = iv;
        aes.Mode = CipherMode.CBC;
        aes.Padding = PaddingMode.PKCS7;
        MemoryStream ms = new MemoryStream();
        CryptoStream cs = new CryptoStream(ms, aes.CreateDecryptor(), CryptoStreamMode.Write);
        cs.Write(payload, 0, payload.Length);
        cs.FlushFinalBlock();
        return ms.ToArray();
    }
    public static byte[] hash(byte[] key) {
        return (new SHA256Managed()).ComputeHash(key);
    }
    public static byte[] dohmac(byte[] key, byte[] data) {
        return (new HMACSHA256(key)).ComputeHash(data);
    }
    public static void Main(string[] args) {
        Console.WriteLine(decryptPayload("SECRET_KEY", "pX0WUfsSJQin8U01dmAHJHyZp1+M+CLnyw2oYS1ty
CchJjSWcOl5oRcHl0LCq9+9YaGyl+WthVfKHRnTf+J7wMCFqcJzr6PXTxP7uKPXaHMrWQyIY5xXGN9Du8np75Em",
"6kH6cPtgvpgS4DvxsMWNg9W3CwN9AmhDFHmNQAjBvR4JBIVl4LznmIXESc9CgE87D51AdqBVacGkR3bnCaBmfKib
/sVQoOQ/uNvBl00Rrqr="));
    }
}
```

Node.js Decryption Example

The following Node.js JavaScript code provides an example of decrypting the payload:

```
var crypto = require('crypto')

function decryptPayload(key, wrappedKey, payload) {
  var unwrappedKey = decrypt(key, wrappedKey);
  return decrypt(unwrappedKey, payload);
}

function decrypt(key, data) {
  var dataBuffer = new Buffer(data, 'base64');
  // TODO: Check that data is at least bigger than HMAC + IV length
  var hmac = dataBuffer.toString('binary', 0, 32);
  var iv = dataBuffer.toString('binary', 32, 48);
  var decodedData = dataBuffer.toString('binary', 48);
  if (hmac !== doHmac(key, iv + decodedData)) {
    // TODO: Handle HMAC validation failure
    return 0;
  }
  var cipher = crypto.createDecipheriv('aes-256-cbc', hash(key), iv);
  return cipher.update(decodedData, 'binary', 'binary') + cipher.final('binary');
}

function hash(data) {
  var hasher = crypto.createHash('sha256');
  hasher.update(data, 'binary');
  return hasher.digest('binary');
}

function doHmac(key, data) {
  var hmac = crypto.createHmac('sha256', key);
  hmac.update(data);
  return hmac.digest('binary');
}
```

PHP Decryption Example

The following PHP code, using PHP version 5.3.0 or later with OpenSSL support, provides an example of decrypting the payload:


```
<?php
function decryptPayload($key, $wrappedKey, $payload) {
    $unwrappedKey = decrypt($key, $wrappedKey);
    return decrypt($unwrappedKey, $payload);
}

function decrypt($key, $data) {
    $decodedData = base64_decode($data);
    // TODO: Check that data is at least bigger than HMAC + IV length
    $hmac = substr($decodedData, 0, 32);
    $iv = substr($decodedData, 32, 16);
    $data = substr($decodedData, 48);
    if ($hmac != hmac($key, $iv . $data)) {
        // TODO: Handle HMAC validation failure
        return 0;
    }
    return openssl_decrypt($data, 'aes-256-cbc', hashKey($key), OPENSSL_RAW_DATA, $iv);
}

function hashKey($data) {
    $hasher = hash_init('sha256');
    hash_update($hasher, $data);
    return hash_final($hasher, true);
}

function hmac($key, $data) {
    return hash_hmac('sha256', $data, $key, true);
}
?>
```

Python Decryption Example

The following Python code, using the M2Crypto wrapper (version 0.21.1) around OpenSSL, provides an example of decrypting the payload:

```
from M2Crypto import EVP
import base64
import hashlib
import hmac

def decryptPayload(key, wrappedKey, payload):
    unwrappedKey = decrypt(key, wrappedKey)
    return decrypt(unwrappedKey, payload)

def decrypt(key, data):
    decodedData = base64.b64decode(data)
    # TODO: Check that data is at least bigger than HMAC + IV length
    hmac = decodedData[0:32]
    iv = decodedData[32:48]
    data = decodedData[48:]
    if hmac != doHmac(key, iv + data):
        # TODO: Handle HMAC validation failure
        return ''
    cipher = EVP.Cipher('aes_256_cbc', hash(key), iv, 0)
    unencrypted = cipher.update(data)
    return unencrypted + cipher.final()

def hash(data):
    hasher = hashlib.sha256()
    hasher.update(data)
    return hasher.digest()

def doHmac(key, data):
    hmacer = hmac.new(key, data, hashlib.sha256)
    return hmacer.digest()
```

Ruby Decryption Example

```
require 'openssl'
require 'base64'

def decryptPayload(key, wrappedKey, payload)
  unwrappedKey = decrypt(key, wrappedKey)
  decrypt(unwrappedKey, payload)
end

def decrypt(key, data)
  decodedData = Base64.strict_decode64(data)
  # TODO: Check that data is at least bigger than IV length
  if (decodedData.byteslice(0,32) !=
      hmac(key,decodedData.byteslice(32,decodedData.bytesize-32)))
    # TODO: Handle HMAC validation failure
    return ''
  end
  cipher = OpenSSL::Cipher.new('AES-256-CBC')
  cipher.decrypt
  cipher.key = hash(key)
  cipher.iv = decodedData.byteslice(32, 48)
  cipher.update(decodedData.byteslice(48, decodedData.bytesize)) + cipher.final
end

def hash(data)
  digest = OpenSSL::Digest::SHA256.new
  digest.update(data)
  digest.digest
end

def hmac(key, data)
  OpenSSL::HMAC.digest(OpenSSL::Digest::SHA256.new, key, data)
end
```

SHA256–Bit Hashing

B

SHA256–Bit Hashing Algorithm

SHA256-bit hashing is required for any string that includes your shared secret, such as the `x-pay-token` header in API calls. These cases are unrelated to the return or decryption of consumer payment information; specifically, SHA256-bit hashing is not used to decrypt payment data. The algorithms used for decryption are different.

The strings to be encrypted are specific to the context; for example, the encrypted string in the `token` field contains different content than encrypted string in the `x-pay-token` header. The SHA256-bit hashing algorithm itself does not change, only the input string to be encrypted. The output from the hash is an encrypted string that is represented in 64 bytes.

Note: *You cannot decrypt a string once it has been encrypted with SHA256-bit hashing. You use the encrypted string for comparison only. If your encrypted string is not the same as the encrypted string created by Visa Checkout, Visa Checkout rejects your request. When Visa Checkout returns a signature in the response, you should create your own string with the same fields separated by ampersands (&) where required, in the same order, and encrypt it for comparison. If your encrypted string does not match the signature, you should not trust that the response came from Visa Checkout.*

Creating a Test x-pay-token Header

You can use the following example to test your SHA-256 hashing algorithm. Consider the following information needed to create an `x-pay-token`, which is hashed together to form the token in an `x-pay-token` header:

- Shared secret associated with the API key:
1234567890123456789012345678901234567890
Use the exact value for your test.
- Time of the request, as a UNIX UTC timestamp:
1419909744
Use the exact value for your test.
- API resource path (name of the endpoint):

onboarding

Use this exact value for your test.

- Request query string:

apikey=abcdefghijklmnopqrstuvwxyzknowmyabcssecretkey

Use this exact value for your test.

- Complete request body:

```
{
  "companyPrimaryLegalName": "My Merchant1",
  "primaryWebsiteURL": "http://www.merchant1.com",
  "relationships": [
    {
      "externalClientId": "MyMerchant1ID"
    }
  ]
}
```

Use this exact value for your test.

Note: A request body does not have line breaks; they are shown here only for readability. The actual request body in this example does not contain white space.

The string to be hashed should look like the following one:

```
1234567890123456789012345678901419909744onboardingapikey=abcdefghijklmnopqrstuvwxyzknowmyabcssecretkey{"companyPrimaryLegalName": "My Merchant1", "primaryWebsiteURL": "http://www.merchant1.com", "relationships": [{"externalClientId": "MyMerchant1ID"}]}
```

Note: The request body does not have line breaks; they are shown here only for readability. The actual request body in this example does not contain white space.

After hashing the fields together using one of the methods shown in , the x-pay-token, which is the hashed result of the sting is as follows:

```
5c4728f2b535aafc0c45b6563ef7fe424593c4cb602698b43705a1de6965c7aa
```

The complete x-pay-token header specifies the transaction timestamp in the x field; thus, the header should look like the following one:

```
x-pay-token: x:1419909744:5c4728f2b535aafc0c45b6563ef7fe424593c4cb602698b43705a1de6965c7aa
```

PHP Example

The following PHP example shows how to create the x-pay-token header:

```
<?php
//Hash for x-pay-token
$time = time();
$callid = "...";
$resource = "payment/data/".$callid;
$apikey = "...";
$query_string = "apikey=".$apikey."&dataLevel=SUMMARY";
$secret = "...";
$token = $secret.$time.$resource.$query_string;
$hashtoken = "x:".$time.":".hash('sha256',$token);
$url = "https://sandbox.secure.checkout.visa.com/wallet-services-web/"
    .$resource."?". $query_string;

//Header
$header = (array("x-pay-token: ".$hashtoken, "Accept: application/json",
    "Content-Type: application/json"));
?>
```

Important:

Because the example uses the shared secret, it should never be run from a web page. Only execute the sample from a protected server.

SHA256–Bit Hashing Examples

SHA256–Bit Hash Java Example

```
import java.security.MessageDigest;
import java.security.SignatureException;
String sourceString = ...; // shared secret + fields in correct format
String hash = sha256Digest(sourceString);

public String sha256Digest (String data) throws SignatureException {
    return getDigest("SHA-256", data, true);
}

private String getDigest(String algorithm, String data, boolean toLower)
    throws SignatureException {
    try {
        MessageDigest mac = MessageDigest.getInstance(algorithm);
        mac.update(data.getBytes("UTF-8"));
        return toLower ?
            new String(toHex(mac.digest())).toLowerCase() : new String(toHex(mac.digest()));
    } catch (Exception e) {
        throw new SignatureException(e);
    }
}

private String toHex(byte[] bytes) {
    BigInteger bi = new BigInteger(1, bytes);
    return String.format("%0" + (bytes.length << 1) + "X", bi);
}
```

Note: *org.apache.commons.code.digest is a useful library for generating the SHA256 hash.*

SHA256–Bit Hash PHP Example

```
$hash = hash('sha256',$sourceString);
```

SHA256–Bit Hash Ruby Example

```
require 'digest'
hash = Digest::SHA256.hexdigest(sourceString)
```

SHA256–Bit Hash Python Example

```
hash = hashlib.sha256(sourceString).hexdigest()
```

SHA256–Bit Hash C# Example

```
import System.Security.Cryptography.SHA256;

public string GetHashSha256(string text)
{
    byte[] bytes = Encoding.ASCII.GetBytes(text);
    SHA256Managed hashstring = new SHA256Managed();
    byte[] hash = hashstring.ComputeHash(bytes);
    string hashString = string.Empty;

    foreach (byte x in hash)
    {
        hashString += String.Format("{0:x2}", x);
    }
    return hashString;
}
```

AVS and CVV Responses

C

AVS Codes

AVS codes can be returned by Visa Checkout in the `avsResponseCode` response field.

AVS Code	Description
0	Unavailable. AVS is not available due to a timeout or other system reason. Note: <i>Although AVS is verified when a card is added to the Visa Checkout account, verification information may not always be available in the consumer information payload; in which case, 0 is returned.</i>
1	Not supported: AVS is not supported for this processor or card type.
2	Unrecognized: the processor returned an unrecognized value for the AVS response.
A	Partial match: street address matches, but 5-digit and 9-digit postal codes do not match.
B	Partial match: street address matches, but postal code is not verified. Returned only for non U.S.-issued Visa cards.
C	No match: street address and postal code do not match. Returned only for non U.S.-issued Visa cards.
D	Match: street address and postal code match. Returned only for non U.S.-issued Visa cards.
E	Invalid: AVS data is invalid or AVS is not allowed for this card type.
F	Partial match: card member's name does not match, but billing postal code matches. Returned only for the American Express card type.
G	Not supported: non-U.S. issuing bank does not support AVS.
H	Partial match: card member's name does not match, but street address and postal code match. Returned only for the American Express card type.
I	No match: address not verified. Returned only for non U.S.-issued Visa cards.
J	Match: card member's name, billing address, and postal code match. Shipping information verified and chargeback protection guaranteed through the Fraud Protection Program. Returned only if you are signed up to use AAV+ with the American Express Phoenix processor.

AVS Code	Description
K	Partial match: card member's name matches, but billing address and billing postal code do not match. Returned only for the American Express card type.
L	Partial match: card member's name and billing postal code match, but billing address does not match. Returned only for the American Express card type.
M	Match: street address and postal code match. Returned only for non U.S.-issued Visa cards.
N	No match: one of the following: Street address and postal code do not match. Card member's name, street address, and postal code do not match. Returned only for the American Express card type
O	Partial match: card member's name and billing address match, but billing postal code does not match. Returned only for the American Express card type.
P	Partial match: postal code matches, but street address not verified. Returned only for non U.S.-issued Visa cards.
Q	Match: card member's name, billing address, and postal code match. Shipping information verified but chargeback protection not guaranteed (Standard program). Returned only if you are signed up to use AAV+ with the American Express Phoenix processor.
R	System unavailable.
S	Not supported: U.S.-issuing bank does not support AVS.
T	Partial match: card member's name does not match, but street address matches. Returned only for the American Express card type.
U	System unavailable: address information unavailable for one of these reasons: The U.S. bank does not support non-U.S. AVS. Or The AVS in a U.S. bank is not functioning properly.
V	Match: card member's name, billing address, and billing postal code match. Returned only for the American Express card type.
W	Partial match: street address does not match, but 9-digit postal code matches.
X	Match: street address and 9-digit postal code match.
Y	Match: street address and 5-digit postal code match.
Z	Partial match: street address does not match, but 5-digit postal code matches.

CVV Codes

CVV codes can be returned by Visa Checkout in the `cvvResponseCode` response field.

CVV Code	Description
0	Unavailable. CVV is not available due to a timeout or other system reason. Note: Although the card security code, e.g. CVV2, is verified when a card is added to the Visa Checkout account, verification information may not always be available in the consumer information payload; in which case, 0 is returned.
1	Card verification is not supported for this processor or card type.
2	An unrecognized result code was returned by the processor for the card verification response.

CVV Code	Description
3	No result code was returned by the processor.
M	The CVN matched.
P	The CVN was not processed by the processor for an unspecified reason.
S	The CVN is on the card but was not included in the request.
U	Card verification is not supported by the issuing bank.
X	Card verification is not supported by the card association.

US, Canadian, and Australian Location Abbreviations

D

Shipping and billing addresses for the United States, Canada, and Australia use standard abbreviations for states, provinces, and other locations.

United States Abbreviations for States and Mailing Locations

State/Location	Abbreviation
ALABAMA	AL
ALASKA	AK
AMERICAN SAMOA	AS
ARIZONA	AZ
ARKANSAS	AR
CALIFORNIA	CA
COLORADO	CO
CONNECTICUT	CT
DELAWARE	DE
DISTRICT OF COLUMBIA	DC
FEDERATED STATES OF MICRONESIA	FM
FLORIDA	FL
GEORGIA	GA
GUAM	GU
HAWAII	HI
IDAHO	ID
ILLINOIS	IL
INDIANA	IN
IOWA	IA

State/Location	Abbreviation
KANSAS	KS
KENTUCKY	KY
LOUISIANA	LA
MAINE	ME
MARSHALL ISLANDS	MH
MARYLAND	MD
MASSACHUSETTS	MA
MICHIGAN	MI
MINNESOTA	MN
MISSISSIPPI	MS
MISSOURI	MO
MONTANA	MT
NEBRASKA	NE
NEVADA	NV
NEW HAMPSHIRE	NH
NEW JERSEY	NJ
NEW MEXICO	NM
NEW YORK	NY
NORTH CAROLINA	NC
NORTH DAKOTA	ND
NORTHERN MARIANA ISLANDS	MP
OHIO	OH
OKLAHOMA	OK
OREGON	OR
PALAU	PW
PENNSYLVANIA	PA
PUERTO RICO	PR
RHODE ISLAND	RI
SOUTH CAROLINA	SC
SOUTH DAKOTA	SD
TENNESSEE	TN
TEXAS	TX
UTAH	UT
VERMONT	VT
VIRGIN ISLANDS	VI
VIRGINIA	VA
WASHINGTON	WA

State/Location	Abbreviation
WEST VIRGINIA	WV
WISCONSIN	WI
WYOMING	WY
US Armed Forces Military Locations	
Armed Forces Africa	AE
Armed Forces Americas (except Canada)	AA
Armed Forces Canada	AE
Armed Forces Europe	AE
Armed Forces Middle East	AE
Armed Forces Pacific	AP

Canadian Province Abbreviations

Province	Abbreviation
Alberta	AB
British Columbia	BC
Manitoba	MB
New Brunswick	NB
Newfoundland and Labrador	NL
Northwest Territories	NT
Nova Scotia	NS
Nunavut	NU
Ontario	ON
Prince Edward Island	PE
Quebec	QC
Saskatchewan	SK
Yukon	YT

Australian State and Territory Abbreviations

State or Territory	Abbreviation
New South Wales	NSW
Australian Capital Territory	ACT
Victoria	VIC
Queensland	QLD
South Australia	SA
Western Australia	WA

State or Territory	Abbreviation
Tasmania	TAS
Northern Territory	NT

SDK Upgrade and Discontinuance Information

E

Unless otherwise noted, all features from previous versions are included in the upgrade version.

The sample app provided with the SDK identifies other build settings that support integration with Swift framework. In case there are any build issues, review the sample app in the SDK.

Installing 4.3 for the first time

- Support for 4.3 is expected to discontinue 9/30/2017.
- Create separate mobile and web merchant profiles.

Upgrading to 4.3 from 4.0

- Support for 4.0 is expected to discontinue 5/31/2017.
- Functionality will include features from 4.0.
- Create separate mobile and web merchant profiles.

Upgrading to 4.3 from 3.7

- Support for 3.7 is expected to discontinue 3/31/2017.
- Functionality will include features from 3.7 and 4.0.

Upgrading to 4.3 from 3.5

- Support for 3.5 is expected to discontinue 1/31/2017.
- Functionality will include features from 3.5, 3.7, and 4.0.
 - Remove VisaCheckoutLibrary.framework from project
 - Add VisaCheckoutFramework.framework to project
 - Ensure deployment target is iOS8 and up
 - Replace importing headers from `#import <VisaCheckoutFramework/VisaCheckout.h>` to `#import <VisaCheckoutFramework/VisaCheckoutFramework.h>`

- Ensure the following:
 - Visacheckout framework in included Embedded Binaries section in the “General” Project setting.
 - Set “Embedded Content Contains Swift Code” to YES in Build Settings.

Upgrading to 4.3 from 3.3

- Support for 3.3 is expected to discontinue 9/20/2016.
- Functionality will include features from 3.3, 3.5, 3.7, and 4.0.
- Upgrade steps:
 - Remove VisaCheckoutLibrary.framework from project
 - Add VisaCheckoutFramework.framework to project
 - Ensure deployment target is iOS8 and up
 - Replace importing headers from `#import <VisaCheckoutFramework/VisaCheckout.h>` to `#import <_VisaCheckoutFramework/VisaCheckoutFramework.h>`
- Ensure the following:
 - Visacheckout framework in included Embedded Binaries section in the “General” Project setting.
 - Set “Embedded Content Contains Swift Code” to YES in Build Settings.

What's New in Prior Releases

F

What's New in Version 4.0

- Two new payment types are supported for Brazil: `ELECTRON` and `ELO`.
- Portuguese language strings are supported for Brazil.
- The first line of a Brazilian address is subdivided into 3 fields within the consumer information payload: street number (`streetNumber`), street name (`streetName`), and additional location information (`additionalLocation`); the complete first line (`line1`) remains available.
- Address auto-complete is also supported.
- A library delegate method, `GAISharedInstance`, is available for use with Google Analytics.

What's New in Version 3.7

Version 3.7 of the SDK provides support for the following new features:

- Apple iPad Tablet Support
Consumers can now use Visa Checkout in merchant-provided iPad apps.
- Expanded Selection of Buttons and Marks
A number of new buttons and marks have been provided, which allow merchants to utilize a wider variety of new designs when integrating the Visa Checkout SDK into their apps. See the Branding Requirements chapter of the *Visa Checkout Getting Started Guide* for more information.
- Enhanced Card on File Support
Merchants have the ability to provide Visa Checkout consumers the option of placing their credit cards on file with the merchant to use as payment on the merchant's app. Enhancements were made to the call-to-action buttons to help consumers understand the flow.
- User Interface and Design Enhancements
Improvements were made to the carousel motion for credit cards and addresses. Changes were made in the location of the camera button to provide better visibility for the consumer. Also, input fields are highlighted in blue to provide users with visibility for more efficient data input.

- Singapore and Malaysia support
Merchants will now have enhancements in form fields, which will help process transactions in Singapore and Malaysia. For Singapore, city defaults to Singapore. For Malaysia, a value for city is required.
 - Messaging Improvements
Updates to provide more consumer-friendly messaging for various scenarios.
- In addition, this version contains minor bug fixes and code optimizations.

What's New in Version 3.5

Version 3.5 of the SDK provides support for the following new features:

- Enhanced Support for Apple Touch ID
Customers can access their Visa Checkout account using Apple's Touch ID, which uses touch authentication.
- Market Expansion – Latin America
Visa Checkout SDK now supports Spanish for countries in the Latin America market, including: Mexico, Argentina, Chile, Colombia, and Peru, and Portuguese in Brazil.
- Language Support – Chinese
Visa Checkout SDK now has Visa Checkout UI language support for displaying Chinese (Simplified, Traditional). Consumers must enter all account information, including names and addresses, in English.
- Messaging Improvements
Updates to provide more consumer-friendly messaging for various scenarios.

In addition, this version contains minor bug fixes and code optimizations.

Document Revision History

G

- Version 2.2, July 8, 2014
- Version 2.3, August 5, 2014
- Version 2.4, September 5, 2014
- Version 2.6, December 8, 2014
- Version 2.7, February 10, 2015
- Version 2.9, May 5, 2015
- Version 3.3, September 9, 2015
- Version 3.5, November 18, 2015
- Version 3.7, March 24, 2016
- Version 4.0, May 27, 2016
- Version 4.3, September 13, 2016